FEATURE_ID: A PROTOTYPE SYSTEM FOR AUTOMATIC IDENTIFICATION OF NC MACHINABLE FEATURES FROM SOLID PART MODELS

by

Jeffrey A. Silkman

B.S. Industrial Engineering,
Kansas State University, 1986

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

P. E. Grosh

Major Professor

TABLE OF CONTENTS

List of Figures iii

List of Tables v

1. INTRODUCTION 1

1.1 Definitions in NC Programming and CAD/CAM 5

2. NC PROGRAM GENERATION 10

2.1 Sequence of Operations 10

2.2 Current Automation in NC Program Generation ... 13

2.3 CAD/CAM Integration in NC Program Generation .. 28

3. DATA COMMUNICATION IN NC CAD/CAM APPPLICATIONS 32

3.1 Database Structure in CAD/CAM Systems 33

3.2 Feature Based Modeling 38

3.3 Centralized Exchange Databases 41

3.4 Automated CAD/CAE Database Translation 45

4. PROJECT INTRODUCTION 48

4.1 FEATURE_ID Project Objective 48

4.2 FEATURE_ID Project Outline 48

5. FEATURE_ID SYSTEM DOCUMENTATION 52

5.1 FEATURE_ID System Methodology 52

5.2 FEATURE_ID System Environment 54

5.3 FEATURE_ID System Limitations 56

5.4 A Hypothetical Application 58

5.5 FEATURE_ID Input Database Format 58

5.6 FEATURE_ID Output Database Format 74

5.7 FEATURE_ID Program Documentation 83

6. RESULTS, CONCLUSIONS AND RECOMMENDATIONS 119

6.1 FEATURE_ID Project Results 119

6.2 Project Conclusions 120

6.3 Recommendations 121

REFERENCES 125

APPENDIX A: Hypothetical Part Example Input Database .. 127

APPENDIX B: Hypothetical Part Blank Example Input Database .. 131

APPENDIX C: Feature Removal Database of Hypothetical Example .. 133

APPENDIX D: FEATURE_ID Program Code Main Program 141

APPENDIX E: FEATURE_ID Program Code Unit PROG 144

APPENDIX F: FEATURE_ID Program Code Unit COMMON 165

APPENDIX G: FEATURE_ID Program Code Unit SORT 170

APPENDIX H: FEATURE_ID Program Code Unit PLOT 179

APPENDIX I: FEATURE_ID Program Code Unit MODEL 182

APPENDIX J: FEATURE_ID Program Code Unit READ 187

APPENDIX K: FEATURE_ID Program Code Unit TOP_FACE ... 189

APPENDIX L: FEATURE_ID Program Code Unit POCKETS 201

APPENDIX M: FEATURE_ID Program Code Unit ARRAYS 221

APPENDIX N: TOP_FACE Feature Identification 242

APPENDIX O: POCKET Feature Identification 245

LIST OF FIGURES

2.1 Developmental Stages of NC Programming 11

2.2 Current Automation in NC Program Generation 14

2.3 CAPP System for NC Cut Planning 18

2.4 Volume Decomposition of a Feature 19

2.5 Machining Cut Planning and Tool Specification ... 21

2.6 Cut Plan Optimization 22

2.7 NC Binary Instruction Code 26

2.8 A Simle M&G Code Program Example 27

2.9 Bridging the "Islands of Automation" 30

3.1 Feature Identification of a Machined Part 44

3.2 PDD Exchange Database in NC CAD/CAM Architecture 47

4.1 FEATURE_ID Project Concentration 50

5.1A Manual CAD to CAM Part Data Translation 53

5.1B Automated CAD to CAM Part Data Translation 55

5.2A - 5.2F Part and Part Blank Examples 57

5.3A Part Blank for Hypothetical Application 59

5.3B Completed Part for Hypothetical Application 59

5.4 Part Design Database Dictionary Structure 61

5.5A Shell Attributes 64

5.5B Face Attributes 66

5.5C Loop Attributes 68

5.5D Edge Attributes 68

5.6A Surface Attributes 71

5.6B Curve Attributes 71

5.7 Data Sharing Structure of Input Database 75

5.8 Feature Based Exchange Database Structure 78

5.9 Features of Removal 80

5.10 Feature Components 82

5.11A FEATURE_ID Program Schematic 103

5.11B Schematic of STORE Program Module 105

5.11C Schematic of SELECT Program Module 107

5.11D Schematic of SET_VIEW Program Module 108

5.11E Schematic of MODEL_IT Program Module 110

5.11F Schematic of RUN Program Module 112

5.11G Schematic of D_FEAT Program Module 116

5.11H Schenatic of PRINT Program Module 118

LIST OF TABLES

3.1 CAD/CAE Database Structures 34

5.1A FEATURE_ID Program and Unit Contents 95

5.1B FEATURE_ID Program and Unit Contents Continued ... 96

1. INTRODUCTION

Computer aided design/Computer aided manufacturing (CAD/CAM) systems designed for use in automated Numerical Control (NC) machining operations are becoming a popular research and development topic in engineering and applied artificial intelligence [1]. The goal of such systems is to automate and link three independent functions involved in NC program preparation. These functions include part design, process planning, and NC program generation.

Within the past ten or so years, automated computer systems which address these three NC programming functions have been developed and utilized. Computer Aided Design (CAD) systems have been developed for part designers, Computer Aided Process Planning (CAPP) systems have been incorporated into process planning departments, and interactive NC program generators are available for use by NC programmers. These systems have provided a tremendous productivity boost throughout the NC program generation process by computerizing traditionally manual functions. As manufacturers have become more dependent on the computer to aid in program development, the demand to integrate CAD, CAPP, and NC program generator systems into one "super" system has become a hot topic. The term CAD/CAM has been coined to describe this integrated system philosophy. CAD represents the computer aided design component systems while

CAM represents computer aided manufacturing component systems such as CAPP and NC program generators.

The principal motive for the development of a CAD/CAM system in the area of NC programming is to eliminate the redundant reentry of information which is a common problem in current stand-alone component system architecture. The principal downfall of current independent CAD, CAPP, and NC program generation systems is that due to their independent development, the database formats which support them differ grossly. This leads to the redundant reentry of part specification data into each system which allows for transcription errors and general inefficiency in the NC programming activity.

Justification for the development of CAD/CAM systems in association with NC programming is based on increased accuracy, application to current automatic manufacturing processes, the reduction of errors through automatic error checking procedures, reduced design turnaround time, uniformity of design features achieved by nonmanual procedures, and reduced dependence on highly skilled and highly paid personnel for routine design and manufacturing tasks [2].

The problems in developing a CAD/CAM system for use in NC program generation are numerous. The majority of

research projects currently being conducted in the area of NC CAD/CAM systems focus on an integral part of the whole CAD/CAM system requirement. Individual project constraints for these systems are placed around one system component. For the CAD system component, research in feature-based modeling systems for use in NC manufacturing CAD data specification is presently the hottest issue. In CAPP system component development, large rule-based expert systems which encompass NC machining theory are the focus of attention. The automated NC program generation component has currently reached a highly developed stage. NC machines are relatively easily programmed and controlled by computer, yet translation programs which interpret computer generated designs and process plans must be developed for automatic integration of NC process controlling capabilities with the rest of the CAD/CAM system.

Although great advances have been made in the area of CAD/CAM system component development, there seems to be a research void in the most fundamental requirement in making the NC CAD/CAM system a reality. This void is in the area of data communication. Development of effective data communication protocol used to link the various components within a CAD/CAM system is without a doubt the most complex issue facing researchers in this area today. Before any real gains can be seen in this area, issues concerning

database formats, standardization, and interfacing new data formats with existing systems must all be thoroughly analysed. Currently, some very large government supported projects are providing much needed organization in the hopes of initiating standardized communication formats for use in CAD/CAM system architecture.

The central focus of this paper is in the area of data communication requirements for use in automated NC program generation. Although the specialized field of NC machining is only one application for which CAD/CAM technology is applicable, problems outlined here do pertain to the CAD/CAM development as a whole. In the introductory chapters in this paper, general background definitions and research information are presented in order to provide a basis of understanding for the developmental problems associated with data communication in CAD/CAM systems used for NC program generation.

The primary work presented in this paper is the development and documentation of a prototype system called FEATURE_ID. FEATURE_ID is a computer based system which is designed to provide a basis for a communication link between the CAD and CAM components of a CAD/CAM system specifically limited to NC program development. The capabilities of this system are limited with respect to the scope of applications

a practical real world communication system would need to handle. However, this research does provide some interesting insight into the challenges associated with automating the communication link between CAD/CAM system components.

1.1 DEFINITIONS IN NC PROGRAMMING AND CAD/CAM

The world of NC programming and CAD/CAM development is full of buzzwords and acronyms. As is already evident in this paper, these acronyms are used freely to identify the many systems, functions, and formats associated with this research area. The following sections provide some fundemental definitions of the topics referred to throughout this paper. In addition, some commentary is provided identifying how each definition is associated with the general context of this paper.

1.1.1 CAD:

Computer Aided Design or Drafting (CAD) represents the merger of computer technology with mechanical drawing. Three functions that can be well accomplished using a CAD system are as follows [2]:

1. Line drawings can be created and stored for future reference.
2. Libraries of common symbols used to create line drawings

can be easily accessed.

3. Dimensioning and plotting functions can be utilized in order to save time.

Modern CAD systems have been developed to handle a complete spectrum of part design functions. In addition to electronic drafting, these systems provide means to incorporate part geometry, material, tolerances, and other required specification parameters into a CAD part database. These systems, sometimes termed Computer Aided Design - Computer Aided Engineering (CAD/CAE) systems, are of perticular interest in this paper.

1.1.2 CAM:

Computer Aided Manufacturing (CAM) represents the merger of computer technology with manufacturing functions. CAM is a general term used to identify systems in which computers are utilized to aid in preparing for and processing of manufacturing operations. CAM systems address five areas of manufacturing processing [2]:

1. Production Programming
2. Manufacturing Engineering
3. Industrial Engineering
4. Facilities Engineering
5. Reliability Engineering

In the context of this paper, the production programming and manufacturing engineering areas of CAM are addressed. In the manufacturing engineering area, computer aided process planning (CAPP) systems are used to define manufacturing sequence and scheduling of NC part production. Once the process plan is derived, NC program generators are used to complete the production programming requirement by producing the NC machine control programs used to manufacture the part. Both of these systems, as well as the NC machine itself are all examples of CAM technology.

1.1.3 CAD/CAM:

CAD/CAM represents the integration of CAD and CAM systems. CAD/CAM utilizes the database created by the designer through computer aided design. This information is automatically transferred to CAM systems for use in manufacturing functions. CAD/CAM systems are designed to integrate the various functions required in NC programming.

1.1.4 CAPP:

Computer Aided Process Planning (CAPP) represents the merger of computer technology with process planning functions. CAPP systems are used to organize a plan for making a product based on current information available.

This information includes such items as the physical capabilities of all machines within the plant, types of tooling and fixtures available, production rates, and product design requirements[3].

Process planning in the NC machining environment is used to provide essential information for generation of the correct manufacturing sequence of a part. Part manufacturing and design data are analyzed in order to specify this sequence. This sequence, coupled with actual part design parameters, is then used to produce the program which controls the movements of the NC machine.

In the context of this paper, providing a means to automatically link CAD design databases with CAPP system input data format is of primary concern. By providing such a means, one phase of CAD/CAM integration for use in NC program generation could be accomplished.

1.1.5 NC:

Numerical Control (NC) represents the merger of computer control technology and machines. Computers are used to directly control and monitor the operation of manufacturing machinery. Numerical codes are entered into the computer controller which controls the machine such that a product is manufactured in accordance with the code. In the context of this paper, NC milling machines are of

primary interest.

1.1.6 NC Program Generators:

NC Program Generators are interactive computer systems which provide aid in the effective translation of part design data into an actual NC program. These systems accept part specification data through interactive input methods and then provide aid in specifying tool motions around a workpiece to produce the part. In this manner, a program is derived for use in NC machine control.

In the context of this paper, providing a link between CAD part designs, CAPP process plans, and NC Program generators is of primary interest. If such a link is provided, a CAD/CAM system for automatically generating NC programs could be developed.

2. NC PROGRAM GENERATION

The basic function of the NC programming activity is to provide the necessary code or program used to control the operation of a particular NC machine. This program must represent the most efficient processing plan for a particular NC machine to produce a desired part. In order to generate an NC program, three developmental stages must be completed. These three functions of NC programming include design, process planning, and NC program generation [Figure 2.1].

2.1 SEQUENCE OF OPERATIONS

2.1.1 Part Design:

The part design function involves the actual specification of a part to be machined. At this primary level, data specifications in terms of a geometric part model representation, part material, and part tolerances are made. This information is stored either in the form of a blueprint drawing, or in a CAD/CAE system database for future use by manufacturing personnel.

2.1.2 Part Process Planning:

NC part process planning involves the derivation of a manufacturing procedure which is followed during the production of a part. In order to develop this procedure,

NC PROGRAMMING FUNCTIONS	DESCRIPTION
DESIGN	Initial specification of part to be machined
PROCESS PLANNING	Derivation of processing plan to follow while producing the part
NC PROGRAM GENERATION	Generation of the NC program according to part design and part process plan

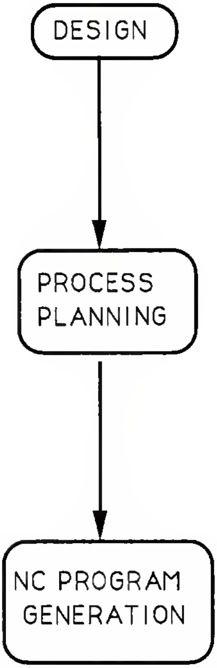


FIGURE 2.1 - DEVELOPMENTAL STAGES OF NC PROGRAMMING

information must be drawn from a part design specification and analyzed in accordance with manufacturing theory. The part design is used to define various machining attributes or features such as holes, pockets and slots of material to be removed from a raw part blank or workpiece. Next, tools are selected for machining the defined features. At this point, tool speed, feed, and depth of cut limits are derived in accordance with the part material specification. Next, consideration of jigs and fixtures must be made in order that the workpiece is properly located and fastened to the NC machine. Finally, a manufacturing precedence is specified to provide a guide for tool motion around the workpiece during machining. Process planning is usually accomplished by experienced manufacturing personnel, but is also done to a limited degree by interactive computer expert systems [1,4].

2.1.3 NC Part Programming:

The NC part programming function involves the production of an actual program or code which is fed into the NC machine computer controller. This code controls the movements of the NC machine in the desired manner in order to produce a part. In order to produce the NC code, an NC part programmer interprets the part design and process plan. From these two inputs, the machine control code is

formulated. Currently, there are interactive NC program development systems which aid the programmer in effectively deriving the code [5].

2.2 CURRENT AUTOMATION IN NC PROGRAM GENERATION

In reference to section 2.1, three general steps are outlined which describe the NC program generation sequence. These are part design, part process planning, and part programming. Traditionally, the elements of these steps are completed manually. Currently, several computer aided systems are available to greatly enhance NC program generation efficiency. This section describes the basic functioning of these systems [Figure 2.2].

2.2.1 Part Design:

Automation in the part design function of NC programming is encompassed by a wide selection of CAD systems. The benefits of using CAD systems for the design of NC machined parts is the ease of creating, editing and storing part representations using a computer. A great deal of development in the CAD system area has already taken place. In order for CAD system databases to effectively represent part data specifications, they must be able to store Product Definition Data (PDD) for later retrieval and use [6]. PDD is the part data specifications in terms of the geometric

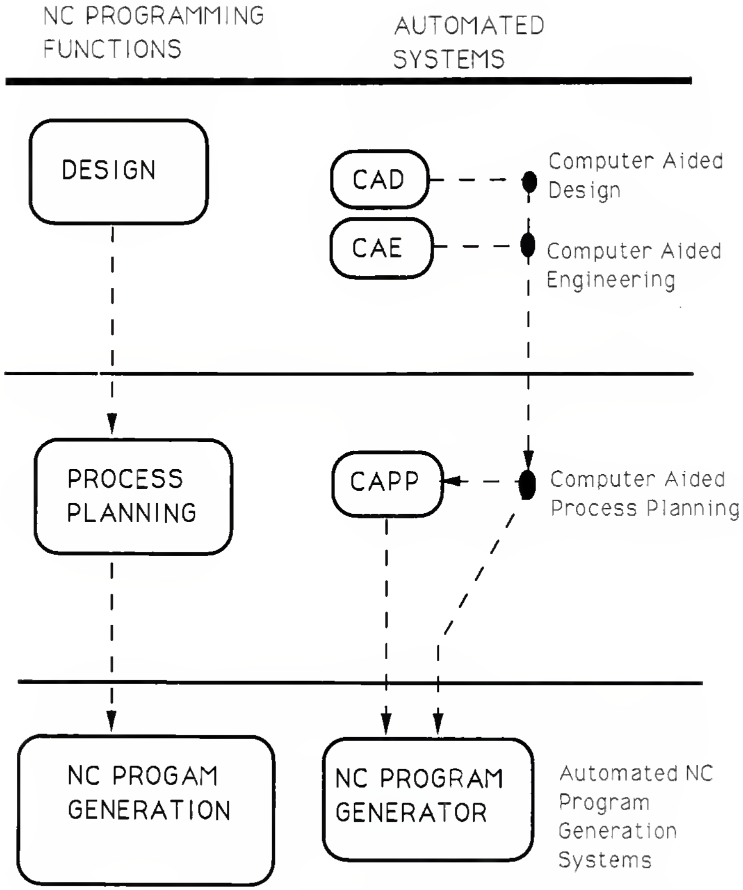


FIGURE 2.2 - CURRENT AUTOMATION IN NC PROGRAM GENERATION

part model, part material, part tolerances, and surface finish to be used by manufacturing personel in the generation of NC programs. Computer Aided Engineering (CAE) systems, which are an advanced form of CAD, are being developed to extend the computers role in the design function by automating the incorporation of PDD into the part database.

In most current NC programming departments, CAD and CAE generated PDD information is utilized by the part process planner and part programmer for analysis and generation of the NC program by visual analysis. Necessary information is visually extracted from the CAD drawing by these personnel and is then used to complete their perspective programming functions. Visual information presented in a CAD drawing is generally in the form of wire frame or solid geometric model graphically represented on a CRT screen. In addition, this model contains textual labels for dimensioning and notes, as well as material specifications for complete part definition.

2.2.2 Process Planning Function:

Computer Automated Process Planning (CAPP) is generally defined as the automatic planning of the manufacturing procedures for producing a product [1]. CAPP systems are designed to create process plans based on all information

about the processing facility and the product description that can be provided. Such information includes the physical capability of the machines in the plant, types of tooling and fixtures available, production rates, and part design requirements. Several CAPP systems are currently being developed and are in limited use specifically in the area of NC program generation. Most of these systems operate via an artificial intellegence based expert computer system. These systems are normally used in areas such as NC process planning where planning decisions are traditionally based on the experience of planning personnel. Knowledge from the expert personnel is placed in a hierarchical computer program which is stepped through to generate a process plan.

CAPP System Architecture -

Computer aided process planning systems formulate a plan by successively decomposing a plan description into a more detailed plan until each action in the plan is a primitive action of the operation plan [1]. This planning technique is known as hierarchical planning. The two essential inputs to this type of system are a part description database and an expert system or rule base from which decisions are made regarding the processing of the

part [Figure 2.3]. The CAPP NC process planning system derives a part production sequence by analysing the part database in accordance with the expert system rule base program. The CAPP control system breaks the planning process into hierarchical levels.

The following breakdown is based on a developmental CAPP system known as XCUT [1]. In the context of this CAPP system, a "feature" refers to a common section of material to be removed from a workpiece in order to manufacture a product. Examples of features include holes, pockets, slots, or notches. These features are identified and input to the CAPP system.

1. Feature Planning:

The first level in developing the operation plan , feature planning, organizes the features identified by the user into a directed graph called a feature access graph. This graph is subjected to a rule base program which determines what order to machine part features [1].

2. Cut Planning:

Cut planning is accomplished by decomposing each machinable feature into machinable volumes [Figure 2.4]. The distinction between machinable features and machinable volumes is that a machinable feature is a convenient

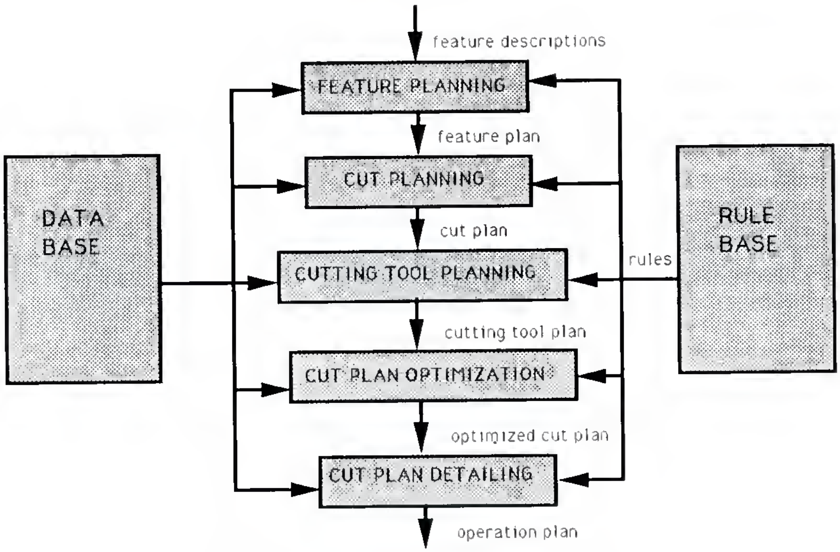


FIGURE 2.3 - CAPP Sytem for NC cut planning [1]

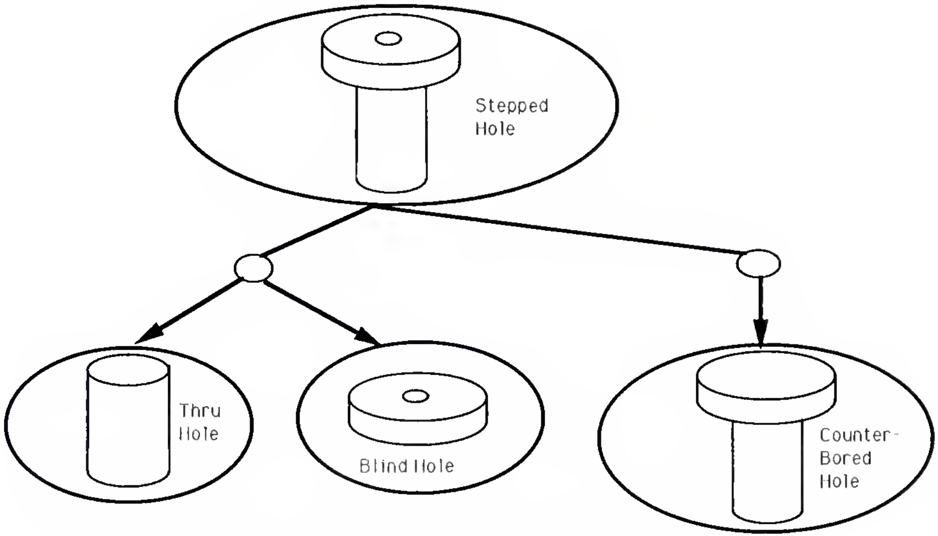


FIGURE 2.4 - Volume Decomposition of a Feature [1]

A compound machinable feature is broken down into basic machinable volumes, or features of removal.

representation of commonly used features by process engineers that might require several machining processes to make. A machinable volume on the other hand, can be made by a single machining process (although it may require several cuts). Once machinable volumes are identified for a particular feature, the cut planning program decomposes these volumes into a set of machining cuts which are performed on the NC milling machine. The cut planning rule base also specifies a tool type to be used for a particular machining cut [1] [Figure 2.5].

3. Cutting Tool Planning:

Cutting tool planning is used to interpret the geometry and tolerance information of each machining cut and specify limits for acceptable values of cutting tool attributes (such as diameter, tool material, flute length, and number of flutes). From these constraints, an ideal tool specification is made for each cut [1].

4. Cut Plan Optimization level:

After all machining cuts have been assigned to a tool list, the plan is optimized in the cut plan optimization planning level. This involves grouping cuts that have intersecting tool lists into the same tool change and ordering the plan in a good machining sequence [1] [Figure 2.6].

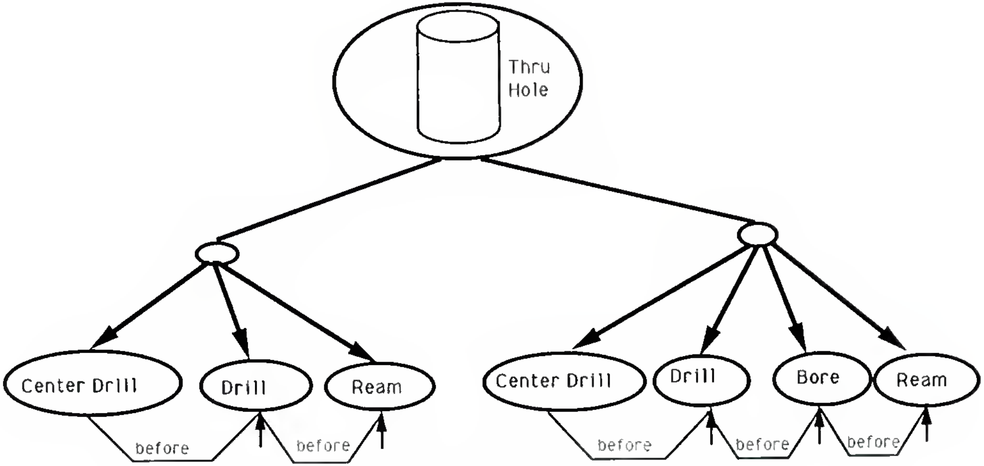


FIGURE 2.5 - Machining Cut Planning and Tool Specification [1].

Cut planning rules decompose machinable volumes into machining cuts based on size and tolerances of the machinable volume.

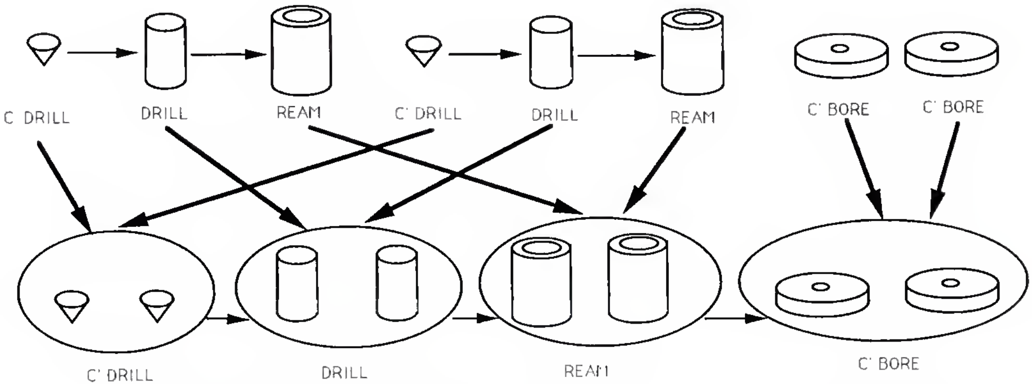


FIGURE 2.6 - Cut Plan Optimiztion [1].

Cut plan optimization groups cuts with intersecting tool lists into the same tool change and orders the cuts into a good machining sequence. This figure shows the initial set of cuts determined by the CAPP system for two identical stepped holes located on a single part. Also shown is the optimized sequence of cuts determined by the CAPP system. In this case, center drilling, drilling, reaming, and counter boring are grouped together to allow for efficient tool changes on the NC machine.

5. Detail Planning:

The last planning level, detail planning, involves specifying speed and feed data for each cut, and reporting the operation plan [1]. In a fully automated NC program generation system, this process plan would be used as input to the NC program generation system [1].

By progressing through these steps, the CAPP system provides a process plan which can be directly interpreted for use in specifying an NC program.

2.2.3 NC Part Programming:

Automated systems are available to the NC part programmer to interactively aid in the effective translation of part design and process plan data into an actual NC part program. Most of these systems are based on a high level NC programming language called Automatically Programmed Tools (APT). The computer system interacts with the part programmer by first accepting geometric information about the part and then aids in the specification of tool motion statements around this geometry. Once an APT program is specified, the system will provide a verification tool path plot simulation for visually checking the program operation. Most of these systems also provide post-processing capabilities in order to automatically produce the lower

level codes required as input for the NC machine CPU [5,7].

NC Machining Operational Sequence -

NC starts with a part programmer who, after studying the engineering drawing and process plan, visualizes the machine operations required to machine the workpiece. The programmer prepares a program by listing codes that define the sequence. A reference point between the workpiece and the machine tool is required. Cutting tools, holding devices, and their location are specified [8]. At this point, tool location and motion path specifications are determined. Once the actual code is determined, it is normally transmitted to the NC machine via a perforated paper or vinyl tape. In more recent applications, magnetic tapes and disks are used for the code storage and transmission. In some cases, codes are transmitted directly to the machines without utilization of a storage medium.

NC Code -

The code required by the NC machine CPU is generally formatted according to the Electronic Industries Association RS 273-A and RS 274-B standard for positioning and controlling [8]. An eight-bit binary coded decimal instruction represents a single input instruction to the NC machine. An NC program line represents an ordered

collection of these instructions [Figure 2.7].

NC Programming Languages -

In order to generate the low level binary code required for NC machine input, higher level programming languages are used for program specification and are subsequently compiled for execution. The most common NC programming language is known as "M&G - Code" [Figure 2.8]. This language is a very structured and mnemonically unfriendly language to work with. However, it is usually required as input to the compiling software in creation of the binary code. Highly user friendly NC programming languages such as Automatically Programmed Tools (APT) are most frequently used in actual NC programming specification. These programs are then "post-processed" by computer software systems into an M&G-Code program which is subsequently compiled into binary instruction codes for NC execution.

Automated NC programming -

Automated NC program generation systems utilize interactive grapics capability to aid in the specification of APT NC programs. Most modern NC programming departments utilize these systems in order to build NC programs. These systems greatly improve the NC programmer's productivity by graphically simulating the workpiece, and the tool

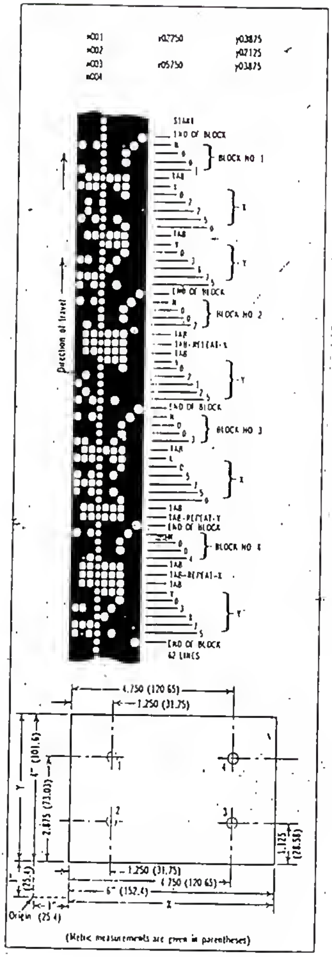


FIGURE 2.7 - NC Binary Instruction Code Stored and Executed Via a Paper Tape.

This is an example of a simple four hole drill pattern specification stored on an NC paper tape. Each line of binary code specifies a specific alphanumeric character which is interpreted by the NC computer control unit. This particular example defines the location of the four holes in terms of sequential moves in the X and Y directions in ten thousandths of an inch.

```
N0010 G01 G91 F0300
N0020 M31
N0030 M09
N0040 G00
N0050 X003125 Y 003125
N0060 G01
N0070 Z-008851
N0080 F0500
```

```
N0010    ==> Statement #10
G01      ==> Use linear interpolation
G91      ==> Move incrementally
F0300    ==> Feed rate code
M31      ==> Rotate spindle at low speed
G00      ==> Point to point positioning
X003125 ==> X value = .3125 inches
Y003125 ==> Y value = .3125 inches
```

FIGURE 2.8 - A Simple M&G Code Program Example.

This example partial program is based on the required program code for a Pratt and Whitney NC machine. This low level language is used to specify tool path, tool speed, and tool feed rates for machining a part

interaction with the workpiece via a dynamic CRT display. The APT programming statements are specified via menu interaction, and are subsequently stored in a line by line format. With these systems it is possible to execute the APT program and visually simulate the NC machining sequence. This allows for complete debugging and verifying of NC programs before actually post-processing, compiling, and executing them on the NC machine.

2.3 CAD/CAM INTEGRATION IN NC PROGRAM GENERATION

2.3.1 CAD/CAM Overview:

The basic methodology required to produce an NC part program involves three hierarchical steps:

1. Design of the part
2. Specificaiton of part process plan
3. Generation of NC program

Within each of these essential programming steps, computer based automated systems are available which aid design and manufacturing personnel in supplying their necessary input to the overall NC program:

1. Design
   - CAD/CAE design systems
2. Process Planning
   - CAPP systems

3. Program Generation

- NC program generation systems

In most cases, each of these automated systems run independently of one another. CAD/CAM represents the integration of these individual yet related systems. The primary basis for CAD/CAM stems from the fact that all of these systems need the same basic input data in order to effectively provide their specific output. CAD/CAM systems store, retrieve, manipulate, and display product definition data - all with unsurpassed speed and accuracy. By using CAD/CAM, engineers are becoming considerably more productive, and product quality and yield are also improved by optimizing energy, materials, and manufacturing personnel [2]. In the NC programming environment, it is easily seen that CAD/CAM integration of the design, process planning, and program generation systems would provide a significant productivity increase due to the elimination of repetitive, routine design and processing functions [2]. NC program generation utilizing today's technology can best be described in terms of "islands of automation" [6]. CAD, CAPP, and NC program generators all operate in a stand-alone environment. CAD/CAM strives to "bridge the islands of automation" by providing communication links between these various automated systems [Figure 2.9].

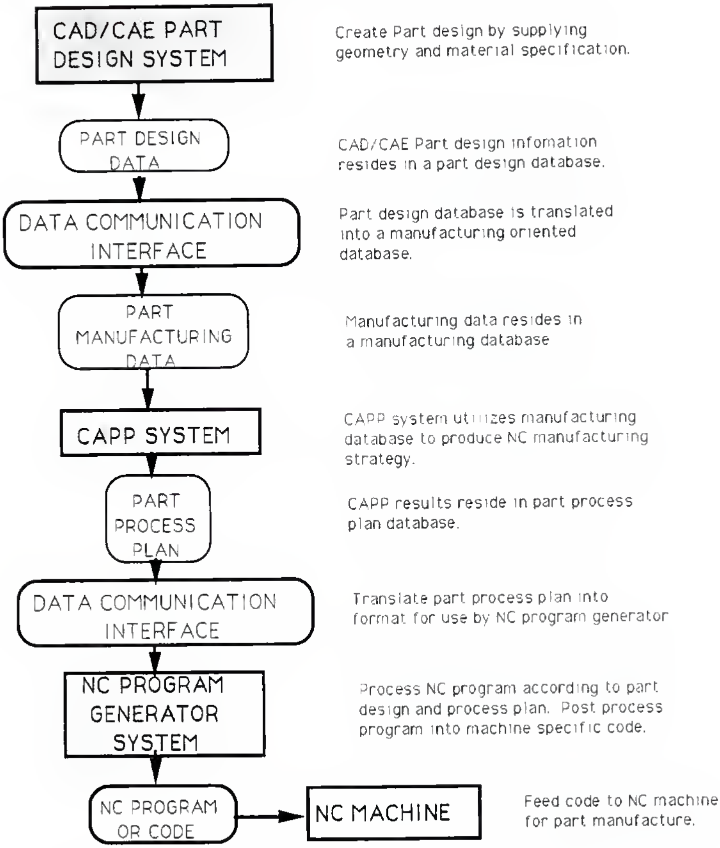


FIGURE 2.9 - Bridging the "Islands of Automation" by providing communication links between stand alone systems.

2.3.2 CAD/CAM Justification in NC Programming:

The CAD/CAM concept has gained quick acceptance in the industrial and design services marketplaces due to the immediate gains in productivity. In addition to direct cost savings, CAD/CAM systems can be justified on the basis of greater accuracy, application to automatic manufacturing processes, the reduction of errors through automatic manufacturing error checking procedures, reduced design turnaround time, uniformity of design quality not achieved through manual procedures, and reduced dependence on highly skilled and highly paid engineers for design. Three major advantages for users of CAD/CAM systems are centralized database creation, data extraction capabilities, and documentation of engineering architectural drawings [2].

Strong demand for CAD/CAM technology is due to four factors. First, productivity is increased from 3- to 10-fold depending on the task to be performed [2]. Second, the lack of trained drafters and technicians is partly compensated through the use of turnkey systems. Third, the CAD/CAM system can produce more complete and better quality designs than existing teams can produce. Fourth, the use of the system eliminates repetitive routine tasks for design and manufacturing personnel [2,9].

3. DATA COMMUNICATION IN NC CAD/CAM APPLICATIONS

In most current NC program development activity, a human interface is required to translate part design specifications produced by CAD/CAE systems and provide the required input to process planning and NC programming functions in order to produce the NC program. Even with such advances as CAD/CAE, CAPP and Automated NC program generation capability, the manual interpretation of a part design is usually required to successfully transfer data between these systems. The philosophy of a true CAD/CAM system for use in NC programming requires that this repetitive manual interpretaion of part data be automated. Only then can the benefits of such a system be realized.

The key to successfully automating communication between the CAD, CAPP, and NC program generation components of CAD/CAM systems is through a centralized computer database. The main function of this database would be to provide common data access for use by all the manufacturing subsystems of an NC CAD/CAM system. In order for this database to provide a true link beween CAD, CAPP, and NC program generators, it must be capable of storing part specification data in a format applicable to automatic computer interpretation. Within this format, design specifications including part geometry as well as manufacturing data would be stored.

Research and development of such standardized databases, sometimes referred to as Product Definition Data (PDD), is currently underway on a large scale. The following sections provide a general overview of the issues involved in this research process and the results and directions currently being pursued in this critical area [Table 3.1].

3.1 DATABASE STRUCTURE IN CAD/CAE SYSTEMS

Most early attempts at developing a centralized PDD database for use in NC CAD/CAM systems started with an analysis of current CAD system database formats. This line of attack seemed like a logical approach to the problem as conventional CAD systems do provide computerized database storage of part specifications. However, what is touted as "computer aided design" in today's automated design applications usually boils down to computer aided drafting [10]. Although CAD system databases have undoubtedly increased the productivity in the design phase of NC part programming by providing draft storage, they provide virtually no useful information required by the subsequent functions of CAPP and NC program generation.

3.1.1 IGES Standard Inadequacy:

In most of today's CAD systems, the Initial Graphics

CAD/CAE System Database	Description
IGES	Initial Graphics Exchange Standard IGES databases store primitive geometric entities used to create a part image on a CRT or plotter
BREP CSG (Geometric Modeling)	Boundary REPresentation Constructive Solid Geometry BREP and CSG part databases store part specifications in terms of the actual geometric and topological data describing a part.
PDD (Geometric Modeling with Manufacturing Data Enhancement)	Product Definition Data PDD databases are enhanced geometric model databases They store vital part descriptive data data such as material and tolerances along with the geometric definition.
FBM	Feature Based Modeling FBM Databases represent a collection of predefined primitive geometric volumes Each volume is given part specific parameters by the designer This collection of volumes represents a part design

Table 3 1 - CAD/CAE DATABASE STRUCTURES

Exchange Standard (IGES) is widely used to define standardized databases for storing the neccessary computer data used in producing a graphical part image, usually on a CRT. These databases store the composition of lines, arcs, and other such basic elements which define the CAD picture. Attempts to use this type of database as a direct interface to CAPP and NC program generators proved fruitless. Simply stated, the IGES database format does not provide part geometry or manufacturing data in a manner which is interpretable by a computer. The sole function of such a database is to store a graphical part image for future visual reference. No product definition data is actually stored. Therfore, a human interpreter is still required to interpret this part image and provide the PDD information to the CAPP and NC program generation systems.

3.1.2 Geometric Modeling:

Once the inadequacies of IGES and similar CAD databases were realized, research focused on the challenge of developing a new CAD database format which would allow for a higher level of part specification storage. In addition to providing the ability to store the image of a part, it was quickly determined that the actual part geometry defining the image must also be saved. This philosophy led to the development of geometric modeling techniques.

The central goal of geometric modeling is to enable the construction of a central database for the information storage, retrieval, and updating of three dimensional mechanical components, assemblies and systems [11]. In essence, the geometric modeling system is an advanced form of CAD which provides the necessary data within its database structure to allow for automatic computer interpretation.

Geometric modeling research has produced an abundance of methods for providing a computer based representation of a part. Of particular interest in this paper is the method of Boundary Representation (BREP). This representation scheme works by storing basic geometrical and topological data such as vectors, points, planes, faces, surfaces, and shells defining a part. In addition, pointers from higher level entities, such as faces, identify which lower level entities, such as points, are associated together. By storing data in this manner, a CAD/CAE system using such a scheme can begin to provide input to the total CAD/CAM system automatically.

With geometric modeling technology, the quest for a centralized database becomes more realistic, but some major inadequacies still linger. Manufacturing information such as part material, tolerances, surface finish requirements, and other key data used by manufacturing systems is not addressed by geometric modeling database structure.

3.1.3 Production Data:

Although geometric modeling provides a geometric and topological description of a part, it does not provide for the storage of manufacturing data related to a part specification. Traditionally this information appears as textual entries on a blueprint or CAD/CAE CRT image. Such descriptions include part materials required for manufacture, machining tolerances, surface finish requirements, heat treating requirements, and other parameters which fully describe a part design. These important pieces of information are needed as input to a process planning system in order to derive the most effective program used by an NC machine to produce a part. Such NC manufacturing parameters as tool type, path, speed, feed rate, and depth of cut are determined not only by part geometry, but also by analysis of the manufacturing data provided by the part designer.

With the realization that manufacturing data is needed in the CAD database, some scattered research has provided methods to accomplish this task. Of particular interest in this paper is the incorporation of manufacturing data into BREP databases. The method is both simple and effective. With this method, allowances for tolerance and material

specifications are appended to the geometric model data. Pointers from higher level entities in the part hierarchy identify which manufacturing data is associated with that entity. This improved database version provides even more potential for automated interfacing of CAD/CAE based designs with the subsequent CAPP and NC program generation functions comprising the total NC CAD/CAM system.

3.2 FEATURE BASED MODELING

Feature Based Modeling (FBM) techniques have been developed to aid the computer in automatically identifying logical sections of a part specification database. In the simplest terms, a feature based model of a part is produced by dividing a part definition database up into small pieces. Each piece represents an individual feature associated with the part. An organized combination of these features defines the part as a whole entity.

A feature refers to common entities such as cubes, holes, bevels, grooves, and other volumes which can be combined or removed in such a way as to define a part. Feature based modeling CAD/CAE systems store the generic format, including geometry and manufacturing data structure, of each feature within its architecture. The part model is built by calling up the feature, defining its parameters,

and then adding it to, or removing it from the database. In essence, a completed feature based part database contains all the information found in a part definition database format describing a complex part, but it is further categorized into simple volumes for easier referencing.

3.2.1 Feature Based Modeling Advantages:

Two major advantages can be derived from feature based modeling methods used within the NC programming activity. First, CAD/CAE systems are improved by simplifying the design specificaiton process. Second, the much sought link between CAD and CAM systems for use in NC program development could be rendered through its use.

CAD systems can be improved tremendously by "packetizing" feature definitions and making them available to the designer while a part specification is being built. Instead of the designer plotting 12 lines to define a rectangular volume, he simply calls to the rectangular volume definition within the FBM - CAD/CAE system structure, provides geometric parameters such as length, width and height, along with manufacturing information such as material composition and tolerance, and then stores the feature in the part database. Similarly, design changes can be easily handled.

Although feature based CAD systems would dramatically

improve the part design phase of the NC programming activity, perhaps the biggest advantage of feature based data base structure is that it could provide a basis for linking CAD with CAM. The basis for this argument is that part features represent simple, solid volumes of material. It would be possible to associate these volumes directly with the machinable volumes commonly defined in NC programs. Therefore, Feature Based Modeling technology could help provide an automated link between design and manufacturing functions within an NC CAD/CAM system.

3.2.2 Feature Based Modeling Methods:

In order to incorporate feature based modeling into CAD/CAE systems, the CAD/CAE system structure must allow for feature representation within the database. Two methods have been investigated throughout current research addressing this problem.

1. Feature Extraction:

The feature extraction method utilizes advanced geometric mathematical algorithms in order to computationally extract features from the CAD part database. This method is designed to analyse IGES-type databases in order that they may be translated into a higher level feature based database format.

2. Design with Features:

The design with features strategy calls for a rather drastic reform in current CAD system structure. With this methodology, a part would be designed by specifying features right from the start. The CAD system would require a feature name, location, and its respective dimensions, tolerances, and other manufacturing information. At this point, an image would be produced on the screen, and the database would be updated. A collection of these features would eventually specify an entire part.

3.3 CENTRALIZED EXCHANGE DATABASES

Although design database formats and methods have improved the ability for computers to store more complete design information, these databases do not neccessarily reflect the data in a format readily usable by manufacturing systems. The primary reason for this discrepancy is based on the contrasting viewpoints of design and manufacturing entities when analysing a part. Design entities think of a part in terms of final specification. This includes such data as part geometry, material, tolerance, and other parameters required to describe a part design. In contrast, NC manufacturing entities think in terms of removing specific volumes of material from a workpiece in order to produce the desired part. The data

which describes a finished part, such as a CAD/CAE produced database, does not directly provide manufacturing systems with descriptions of these volumes of material to be removed. Therefore, an exchange database format capable of handling these material removal definitions is required to bridge CAD and CAM systems. Research in this area has led to the preliminary development of Product Definition Data (PDD) exchange databases which are designed to provide part descriptions in a format directly usable by NC manufacturing functions.

3.3.1 Product Definition Data Exchange Databases:

A standard format for product definition data exchange databases has recently been released. This database format, called Product Definition Exchange Standard (PDES) reflects a part design in terms of manufacturing information used in producing the part.

From the manufacturing perspective, a part design is used to define various volumes of material to be removed from a part blank or workpiece by an NC machine. With today's technology, a manual interpretation of part design data is required to determine these machinable volumes according to the part specification and the initial part blank specification. The data accociated with these volumes is then used as input to CAPP and NC program generator CAM

systems to produce the NC machine control program. In order to automate this process, a database format capable of storing part information in terms of logical material removal volumes is required.

The development of a centralized PDD exchange database would provide this information exchange buffer between CAD and CAM sytems by storing the specific data associated with the volumes to be removed according to a specific part design and a specific part blank. As mentioned in section 3.2.1, Feature Based Modeling techniques are directly applicable to this type of database format. By associating the material removal volumes as features, a database structure which is directly usable by CAM functions could be developed [Figure 3.1].

3.3.2 PDD Exchange Database Format:

The goal of the PDD exchange database format is to supply part specification data in terms of manufacturing operations. In very general terms, the format is constructed by first identifying all of the machinable volumes to be removed from a part blank in order to produce a part. Once this is done, individual specification data must be associated with each volume. This data includes

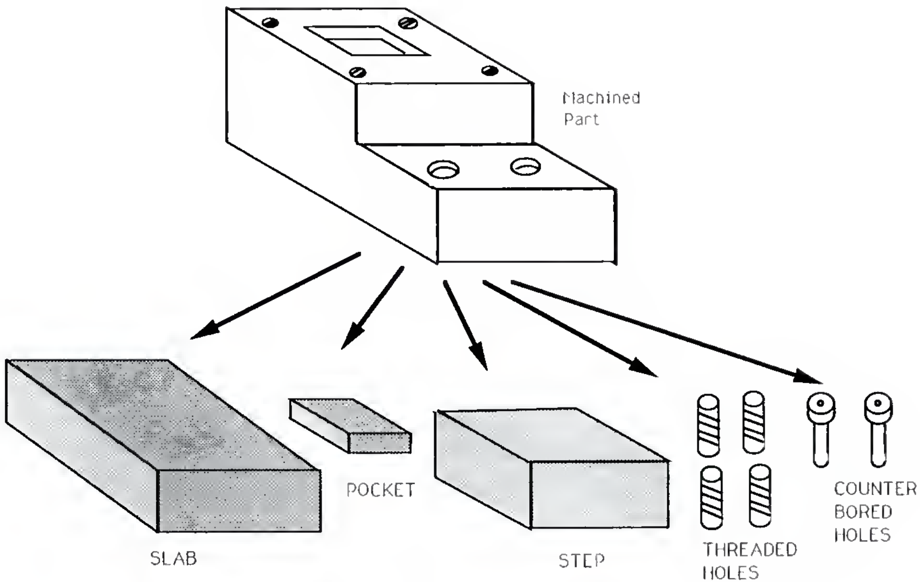


FIGURE 3.1 - Feature Identification of a Machined Part

volume geometry, topology and manufacturing data. A collection of these volume specifications defines the original part design in terms of entities identifiable by manufactuing functions.

Ultimately, this centralized database provides a storage center for the exchange of information between CAD/CAE design functions and CAM manufacturing functions. With such a data communication entity, the link between CAD and CAM can be a reality.

3.4 AUTOMATED CAD/CAE DATABASE TRANSLATION:

The required format for the centralized PDD exchange database has been researched extensively. An American National Standard format for this type of database called Product Definition Exchange Standard (PDES) has been developed [6]. However, there has been only minimal developmental research into the support software which automatically translates CAD/CAE part design databases into these exchange format databases. In order to completely automate data transfer between all of the subsystems of the CAD/CAM system these support programs or translators must be developed.

3.4.1 Automated Translator Requirements:

The function of an automated data translator for use in CAD/CAM system linking would be to provide interpretive

algorithms which accept CAD/CAE part databases as input and produce the PDD exchange database as output. By fully developing such capability, an automated CAD/CAM system for use in NC program generation could be developed.

The basic requirements of such a translation system could be divided into three broad categories. First, the CAD/CAE part database structure would have to be standardized to allow for uniform input data format. Second, the structure of the PDD exchange database would have to be standardized to allow for uniform output data format. Third, the actual algorithms which interpret the input part specification data and derive the PDD exchange output data would have to be developed. By analysing these problem areas, a basic translator system philosophy could be developed for use in NC manufacturing applications as well as all other potential CAD/CAM system functions [Figure 3.2].

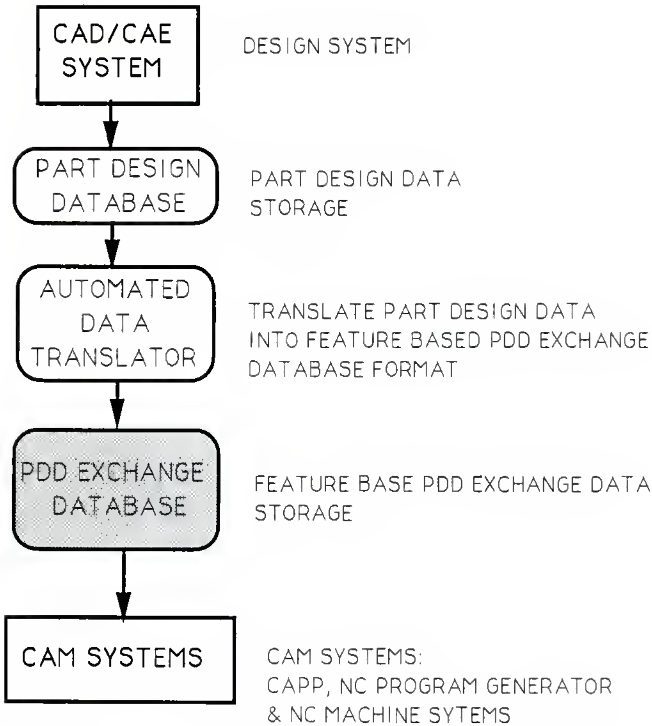


FIGURE 3.2 - PDD EXCHANGE DATABASE IN NC CAD/CAM ARCHITECTURE

4. PROJECT INTRODUCTION

The focus of this paper is on the development of a prototype system called FEATURE-ID. The FEATURE-ID system is designed to analyze automatically CAD/CAE part design and blank design databases and provide a feature based PDD exchange database as output. The principle drive for this project is to understand better what issues must be considered in order to link CAD/CAE systems with CAM systems through effective data communication. The following sections of this paper outline the developmental phases and provide documentation of the FEATURE-ID system.

4.1 FEATURE-ID PROJECT OBJECTIVE:

The objective of the FEATURE-ID project is to develop a computer based system capable of automatically translating a CAD/CAE part design database into a feature based PDD exchange database for use by CAM systems designed to produce NC machine control programs.

It is important to point out that this project does not approach the entire NC CAD/CAM issue, but instead concentrates on the data communication aspects of the issue. Data communication is one of the weakest areas in terms of development within CAD/CAM system architecture.

4.2 FEATURE-ID PROJECT OUTLINE

This section is presented to outline the preliminary as

well as developmental stages of the FEATURE-ID system project. Documentation of the FEATURE-ID system appears in chapter 5.

Project Needs Analysis:

This preliminary phase of the project includes background research and analysis of the needs in developing a system to be used for automated NC program development. The results of this analysis appears in the introductory chapters 1, 2, and 3 in this paper. The emphasis on this stage was to research current technologies applicable to the area and identify what work is needed to link these technologies into an integrated system capable of automatically producing NC programs directly from part designs. The magnitude of work involved in developing such a system is tremendous. Thus, it was decided to concentrate on the data communication issues within the confines of the overall system.[Figure 4.1] Effective solutions to data communication problems are vital to the success of building a system capable of automatically generating NC programs. FEATURE-ID addresses this issue and presents some preliminary solutions to these problems.

The FEATURE-ID project followed three phases of analysis and development:

1. CAD/CAE Database Analysis

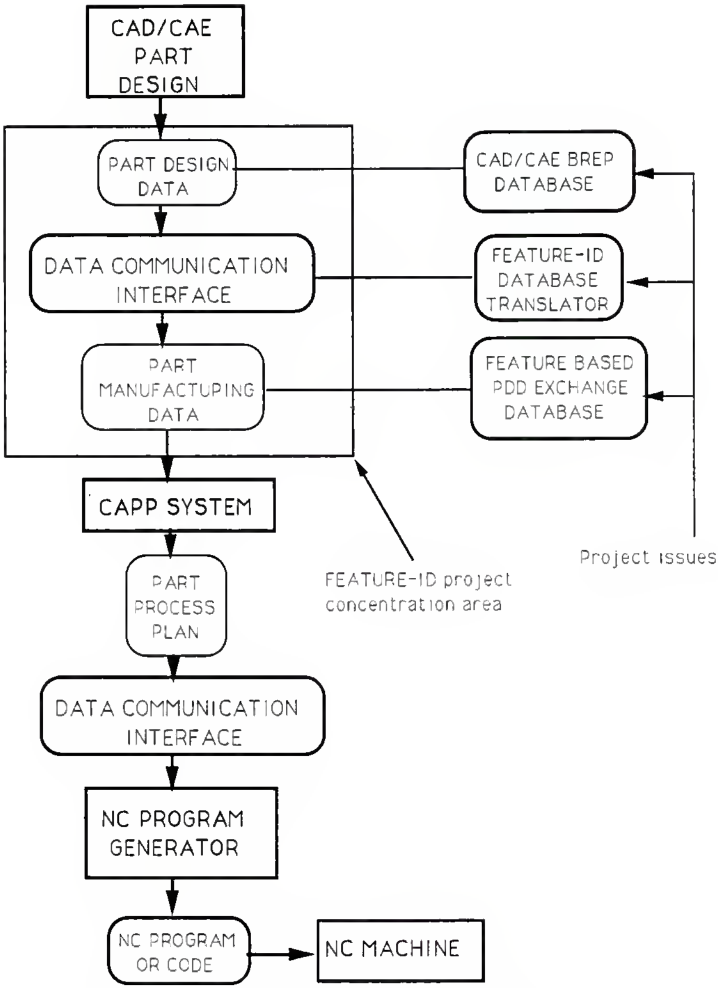


FIGURE 4.1 - FEATURE ID PROJECT CONCENTRATION

2. Feature Based PDD Exchange Database Analysis

3. FEATURE-ID System Development

CAD/CAE Database Analysis:

This primary project phase focuses on CAD/CAE input database formats. In order to develop a system which translates these databases, it was first necessary to understand how they are formatted. Formal documentation of this analysis is presented in chapter 5.

Feature Based PDD Exchange Database Analysis:

This project phase focuses on Feature Based PDD Exchange database formats. In order to develop a system which outputs these databases, it was first necessary to understand how they are formatted. Formal documentation of this analysis is presented in chapter 5.

FEATURE-ID System Development:

Once input and output data formats were analysed, actual system design could progress. The goal of this system is to provide an automatic interpretation of part design data and part blank design data resulting in an output of specific material removal information directly usable by CAM systems for developing an NC program. Documentation of the FEATURE-ID system appears in chapter 5.

5. FEATURE-ID SYSTEM DOCUMENTATION

5.1 FEATURE-ID SYSTEM METHODOLOGY

The goal of the FEATURE-ID system is to automate the translation of CAD/CAE part design data into an exchange format usable by CAM systems which produce NC programs. In order to accomplish this task, it was decided to incorporate traditional manual CAD to CAM data translation methods into a computer program. By outlining these methods, the basic functioning of the FEATURE-ID program can be understood.

In the primary stage of NC program development, a part design specification is made. Once the final design is completed, it becomes the responsibility of manufacturing personnel to determine how to utilize an NC machine to produce the part. The initial step taken by the manufacturing engineer is to specify a raw blank of material from which the part will be produced. This part blank, or workpiece, may be in the form of a solid piece of material or a semi-finished casting. The volumes of material to be removed from the part blank are then determined by manually analysing the part blank geometry subject to the actual part gemoetry. The data derived from this analysis is then used as input to CAM systems [Figure 5.1A]. For NC program generation, CAPP and a NC program generator CAM system can be utilized to formulate the NC program based on the volumes of material to be removed from the part blank.

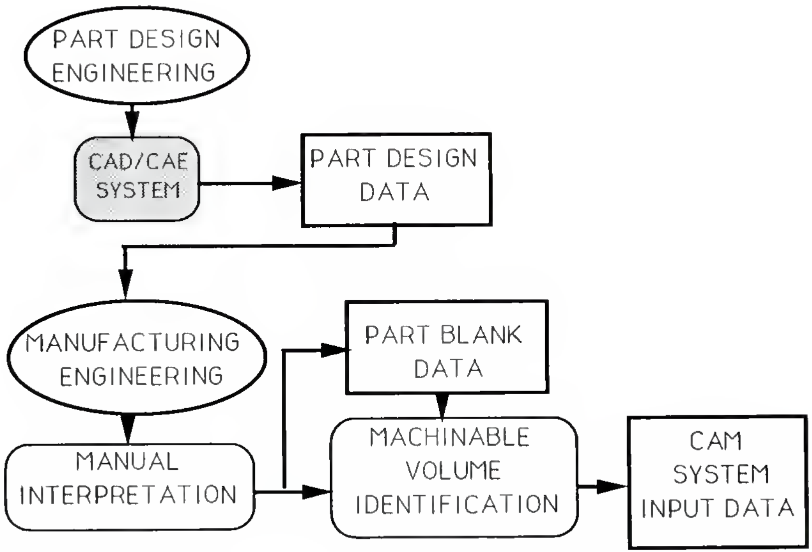


FIGURE 5.1A - MANUAL CAD TO CAM PART DATA TRANSLATION

The FEATURE-ID system incorporates the same basic methodology outlined above within its program structure. First, the system accepts a part specification database as input. Next, a similarly formatted part blank specification database is also received as input. At this point, the program algorithmically extracts the specifications of the volumes of material to be removed from the part blank in order to produce the actual part. The output of the system is a database containing the resulting collection of volumes of material to be removed from the part blank along with their individual geometric and topological data specifications. This output exchange database is the final product of the FEATURE-ID system [Figure 5.1B].

5.2 FEATURE-ID SYSTEM ENVIRONMENT:

The FEATURE-ID system is designed for use with the IBM PC/XT personal computer system. The initial development machine contains an 8088-2 based CPU with 640k RAM. The data storage medium is a 30 megabyte hard drive of which only 1 megabyte or so is actually utilized. A Sysdyne EGA color graphics monitor with 640 by 320 pixel resolution is used for the program interactive textual and graphic support. Also, any IBM-compatible dot matrix printer can be used for producing hard copies of input and output databases.

The software used in developing the FEATURE-ID system

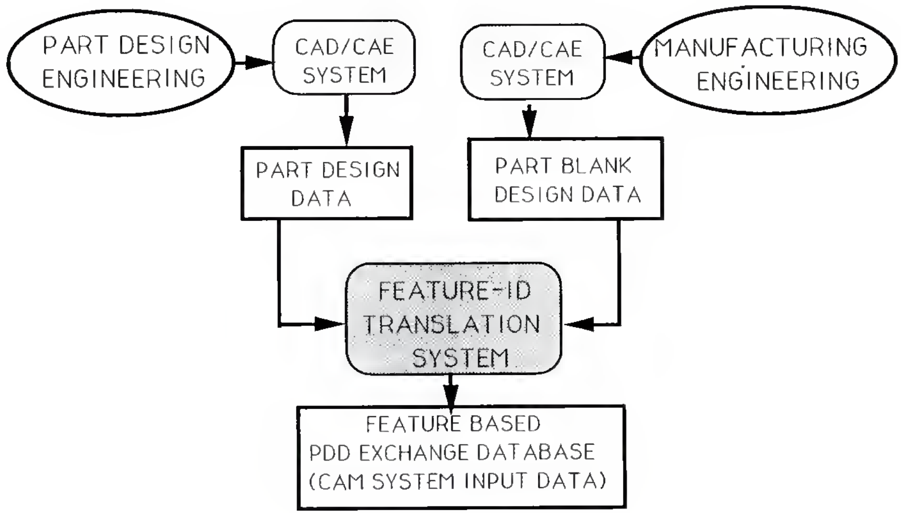


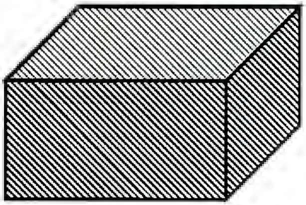
FIGURE 5.1 B - AUTOMATED CAD TO CAM PART DATA TRANSLATION

program is Turbo Pascal 4.0 by Borland International. Turbo Pascal is used exclusively for all program coding. This includes standard Pascal language elements plus specific Turbo Pascal utilities and graphics functions [12]. Also, Wordstar 3.3 by Micropro International can be used for creating and editing input databases as well as the Turbo Pascal code.

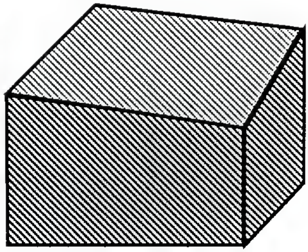
5.3 FEATURE-ID SYSTEM LIMITATIONS:

The FEATURE-ID system is not a complete CAD/CAM system for use in NC program generation. It is a research prototype system which addresses the issue of automatic CAD to CAM database translation. In terms of actual applications, the FEATURE-ID system can handle part designs which can be produced by machining face and pocket volumes from convex polyhedron part blanks [Figure 5.2A, 5.2B]. These volumes of material to be removed, known as part features, can be automatically extracted from the part and part blank database inputs by the system [FIGURE 5.2C - 5.2F].

Due to RAM size of the PC used for this project, the physical size of a part model which can be handled is limited. The maximum number of vertices per part handled is 50. With a larger CPU, the program could be upgraded to accomodate larger models very easily.



5.2A - PART BLANK EXAMPLE
(RECTANGULAR POLYHEDRON)



5.2B - PART BLANK EXAMPLE
(RECTANGULAR POLYHEDRON WITH
NON PARALLEL TOP FACE)

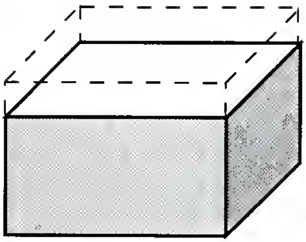


FIGURE 5.2C - PART EXAMPLE
(TOP FACE FEATURE REMOVAL)

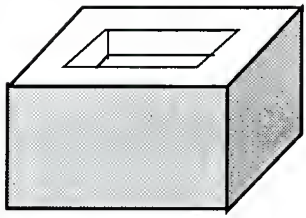


FIGURE 5.2D - PART EXAMPLE
(POCKET FEATURE REMOVED)

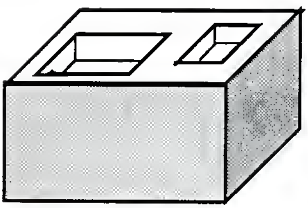


FIGURE 5.2E - PART EXAMPLE
(MULTIPLE POCKET FEATURES)

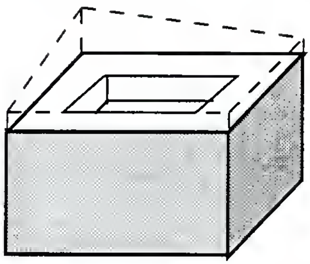


FIGURE 5.2F - PART EXAMPLE
(POCKET & NON-PARALLEL
TOP-FACE)

In terms of database content, the FEATURE-ID system can handle part and part blank geometric and topological specifications. Part manufacturing data such as tolerances and material are not handled by the current system.

It is important to note that the FEATURE-ID system was developed with the intent of providing a methodology for the design of an automatic CAD to CAM database translator. Although the capability of the system is limited, it does provide an initial expandable structure for furthur development in this important area.

5.4 A HYPOTHETICAL APPLICATION

In order to present the the documentation of the FEATURE-ID system in an effective manner, a hypothetical part and part blank will be used as reference [Figure 5.3A, 5.3B]. The input and output databases as well as the program walkthrough will be described in reference to this hypothetical application.

5.5 FEATURE-ID INPUT DATABASE FORMAT

As discussed in chapter 3, section 1.2, the Boundary Representation (BREP) technique is an effective method for geometrically modeling part designs. The part design and part blank design databases produced by using BREP modeling methods are required as input to the FEATURE-ID program. The specific BREP format chosen for use by the FEATURE-ID

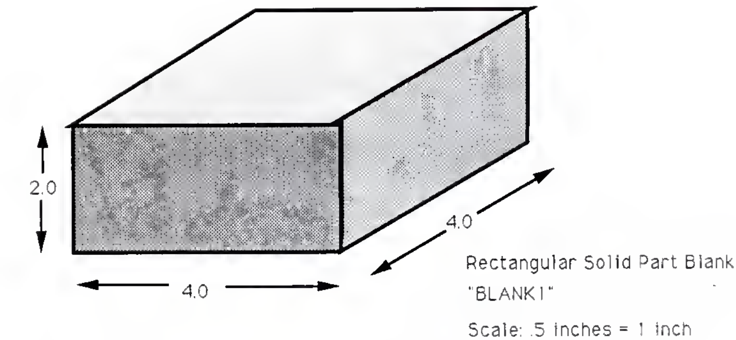


FIGURE 5.3A - PART BLANK FOR HYPOTHETICAL APPLICATION

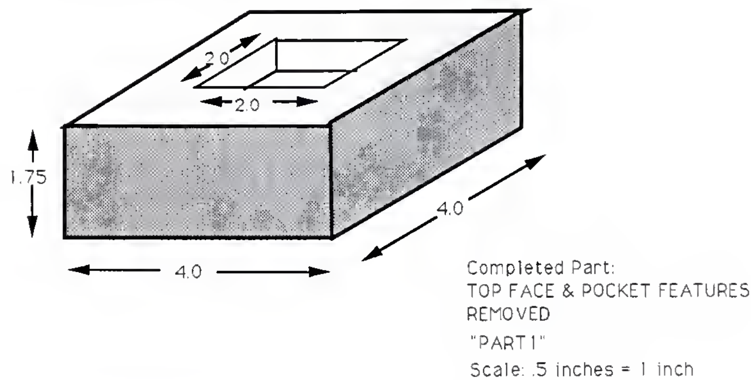


FIGURE 5.3B - COMPLETED PART FOR HYPOTHETICAL APPLICATION

system is taken from the Automated Manufacturing Research Facility (AMRF) topology and geometry part model database standard of 1987 [13,14].

5.5.1 Input Database Dictionary Structure:

The part and part blank design databases are stored on the PC system hard drive as text files. These files are formatted using structured statements. Each line of the text file represents one complete statement. Statement lines can be of two general types:

1. Dictionary Identifiers
2. Dictionary Attributes

A dictionary is a group of related data within the whole database file structure. Dictionary identifier statements mark the beginning and ending of a group of this related data. The data itself is contained in dictionary attribute statements. Each dictionary attribute statement defines a particular entity within each dictionary group. Dictionaries are organized hierarchically within the database structure [Figure 5.4]. This structure is outlined below.

1. Part Model Dictionary -

The level-one Part dictionary encompasses the entire part or part blank database. The first line of the part or

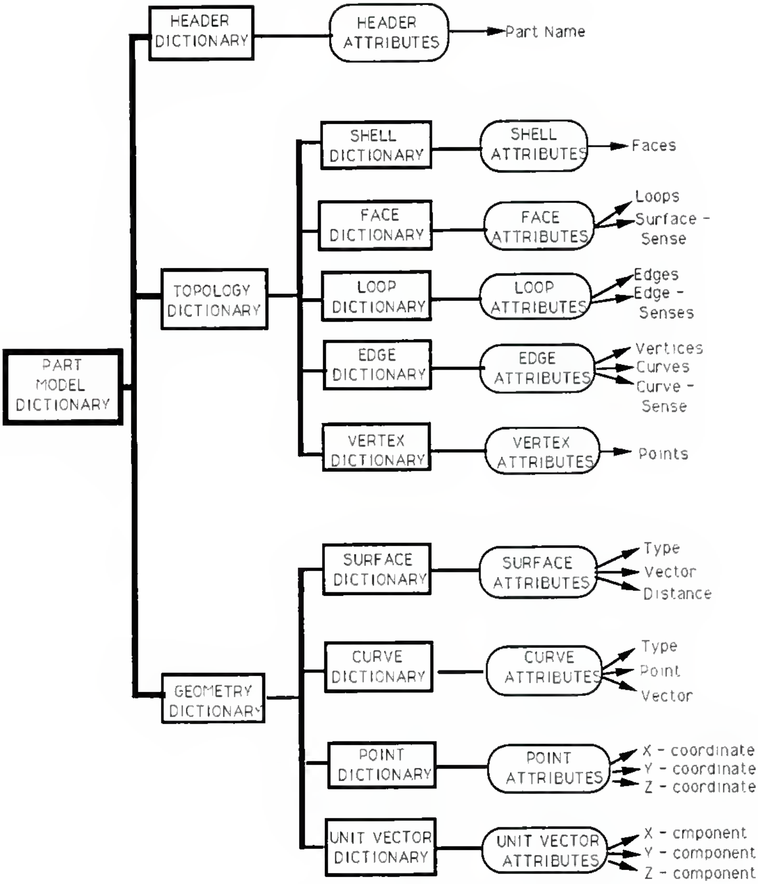


FIGURE 5.4 - PART & PART BLANK DESIGN DATABASE DICTIONARY STRUCTURE REQUIRED FOR FEATURE-ID SYSTEM INPUT.

part bank design input file contains the part dictionary "begin" identifier statement and the last line contains the part dictionary "end" identifier statement. This level-one dictionary contains three level-two dictionaries. These are the Header dictionary, Topology dictionary, and Geometry dictionary.

Part Dictionary Syntax:

```
Identifier:  /PART_MODEL
             <Header Dictionary>
             <Topology Dictionary>
             <Geometry Dictionary>
Identifier:  /END_PART_MODEL
```

2. Header Dictionary -

The level-two Header dictionary contains the identification name assigned to the part or part blank design database.

Header Dictionary Syntax:

```
Identifier:  /HEADER
Attribute:    PART NAME = 'part name'.
Identifier:  /END HEADER
```

3. Topology Dictionary -

The level-two Topology dictionary contains five level-three dictionaries. These dictionaries include the Shells dictionary, Faces dictionary, Loops dictionary, Edges

dictionary, and Vertices dictionary. These dictionaries contain all topological information describing the part or part blank design.

Topology Dictionary Syntax:

```
    Identifier:  /TOPOLOGY
                 <Shells Dictionary>
                 <Faces Dictionary>
                 <Loops Dictionary>
                 <Edges Dictionary>
                 <Vertces Dictionary>
    Identifier:  /END_TOPOLOGY
```

4. Shells Dictionary -

The level-three Shells dictionary contains all of the SHELL definitions associated with a part's topology. Each SHELL definition contains a collection of FACE attributes which define the topological surface or shell of the part [Figure 5.5A].

Shell Dictionary Syntax:

```
    Identifier:  /SHELLS
    Attribute:     SHL###  ; FAC###, FAC###, FAC### .
    Identifier:  /END_SHELLS

    - where ### ==> ID number of entity
            SHL ==> SHELL
            FAC ==> FACE
```

5. Faces Dictionary -

The level-three Faces dictionary contains all of the FACE definitions associated with a part's topology. Each

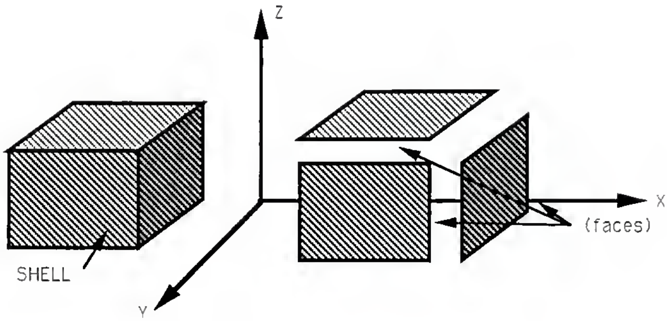


FIGURE 5.5A - SHELL ATTRIBUTES

FACE definition is characterized by one or more LOOP attributes, a SURFACE attribute and a SURFACE positional SENSE attribute. These attributes define the geometric location of each FACE, and also provide a reference to how the solid part material lies with regard to the face position [Figure 5.5B].

Faces Dictionary Syntax:

```
    Identifier:  /FACES
    Attribute:     FAC### ; LOP###, LOP###; SUR### s .
    Identifier:  /END_FACES

    - where ### ==> ID number of entity
            FAC ==> FACE
            LOP ==> LOOP
            SUR ==> SURFACE
              s ==> + or - positional SENSE of part to
                    SURFACE
```

6. Loops Dictionary -

The level-three Loops dictionary contains all of the LOOP definitions associated with a part's topology. Each LOOP definition is characterized by at least three EDGE attributes with one directional SENSE attribute for each EDGE. These attributes are used to define each LOOP by associating individual EDGE definitions together. The EDGE directional SENSE attribute is set according to the clockwise or counter-clockwise direction of the LOOP definition. If the originally defined EDGE direction does not oppose the LOOP direction, EDGE SENSE is positve,

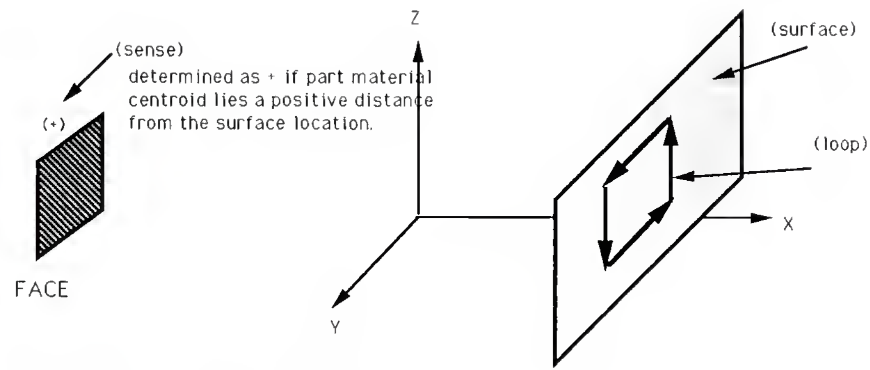


FIGURE 5.5B - FACE ATTRIBUTES

otherwise it is negative [Figure 5.5C].

Loops Dictionary Syntax:

```
Identifier:  /LOOPS
Attribute:     LOP### ; EDG### s , EDG### s , EDG### s .
Identifier:  /END_LOOPS

- where ### ==> ID number of entity
        LOP ==> LOOP
        EDG ==> EDGE
          s ==> EDGE directional SENSE within LOOP
```

7. Edges Dictionary -

The level-three Edges dictionary contains all of the EDGE definitions associated with a part's topology. Each EDGE definition is characterized by two VERTEX attributes, one CURVE attribute and directional SENSE attribute assigned to the EDGE. These attributes are used to define each EDGE according to a line segment geometry associated with the part design [Figure 5.5D].

Edge Dictionary Syntax:

```
Identifier:  /EDGES
Attribute:     EDG### ; VTX### , VTX### ; CRV### s .
Identifier:  /END_EDGES

- where ### ==> ID number of entity
        EDG ==> EDGE
        VTX ==> VERTEX
        CRV ==> CURV
          s ==> directional SENSE assigned to EDGE
```

8. Vertices Dictionary -

The level-three Vertices dictionary contains all of the VERTEX definitions associated with a part's topology. Each

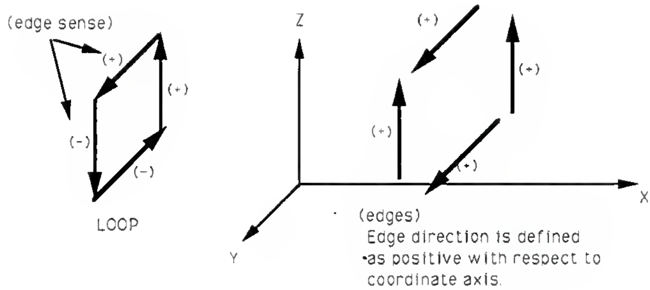


FIGURE 5.5C - LOOP ATTRIBUTES

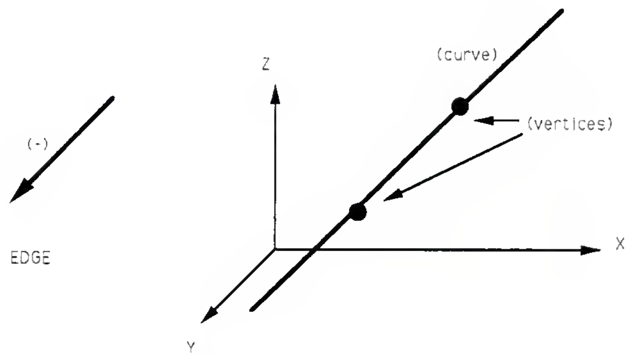


FIGURE 5.5D - EDGE ATTRIBUTES

VERTEX definition contains one POINT attribute. This attribute is used to define the VERTEX based on its coordinate location in the part design geometry.

Vertces Dictionary Syntax:

```
Identifier:  /VERTICES
Attribute:     VTX### ; PNT###.
Identifier:  /END_VERTICES

- where ### ==> ID number of entity
        VTX ==> VERTEX
        PNT ==> POINT
```

9. Geometry Dictionary -

The level-two Geometry Dictionary contains four level-three dictionaries. These dictionaries include the Surfaces dictionary, Curves dictionary, Point dictionary, and Unit Vector dictionary. These dictionaries contain all geometric information describing the part or part blank design.

Geometry Dictionary Syntax:

```
Identifier:  /GEOMETRY
             <Surfaces Dictionary>
             <Curves Dictionary>
             <Points Dictionary>
             <Unit Vectors Dictionary>
Identifier:  /END_GEOMETRY
```

10. Surfaces Dictionary -

The level-three Surfaces dictionary containes all of the SURFACE definitions associated with a part's geometry.

Each SURFACE definition contains a SURFACE TYPE attribute, a UNIT VECTOR attribute, and a DISTANCE from origin attribute. Surface TYPE attributes are limited in this research to linear planes. The UNIT VECTOR attribute defines the normal vector of each SURFACE PLANE definition. The DISTANCE attribute defines the loction of the SURFACE PLANE in relation to the coordinate origin [Figure 5.6A].

Surfaces Dictionary Syntax:

```
    Identifier:  /SURFACES
    Atttribute:    SUR### ; TYPE ; VEC### ; DISTANCE .
    Identifier:  /END_SURFACES

    - where ### ==> ID number of entity
            SUR ==> SURFACE
            TYPE ==> SURFACE TYPE - PLANE only
            VEC ==> UNIT VECTOR normal to plane
            DISTANCE ==> DISTANCE from origin to plane along
                         normal
```

11. Curves Dictionary -

The Curves dictionary contains all of the CURVE definitions associated with a part's geometry. Each CURVE definition is characterized by a CURVE TYPE attribute, a POINT attribute, and a UNIT VECTOR attribute. The CURVE TYPE attribute is limited to LINES in this research. Each CURVE LINE is defined by a POINT coordinate, and a parallel UNIT VECTOR specification [Figure 5.6B].

Curves Dictionary Syntax:

```
    Identifier:  /CURVES
```

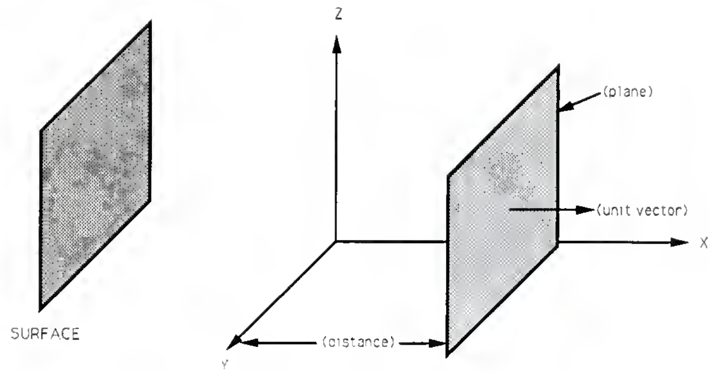


FIGURE 5.6A - SURFACE ATTRIBUTES

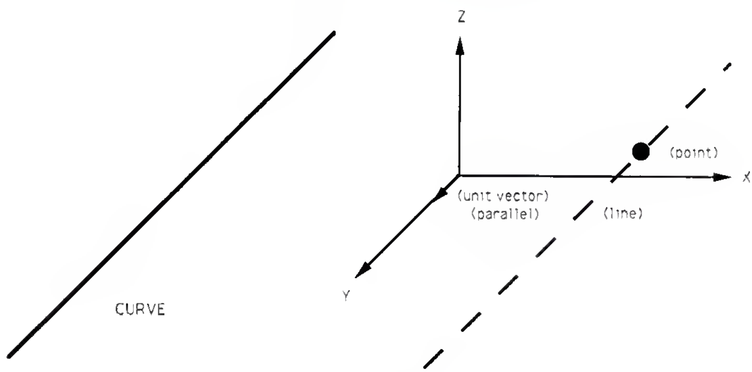


FIGURE 5.6B - CURVE ATTRIBUTES

```
Attribute:     CRV### ; TYPE ; PNT###, VEC### .
Identifier:  /END_CURVES

- where ### ==> ID number of entity
        CRV ==> CURVE
        TYPE ==> CURVE TYPE - LINE in this research
        PNT ==> POINT intersecting CURVE
        VEC ==> UNIT VECTOR parallel to line
```

12. Point Dictionary -

The Point dictionary contains all of the POINT definitions associated with a part's geometry. Each POINT definition is characterized by three real coordinate value attributes. These coordinate values refer to the X, Y, and Z coordinates of a POINT contained in a part's geometry.

Point Dictionary Syntax:

```
Identifier: /POINTS
Attribute:     PNT### ; X-COORD, Y-COORD, Z-COORD .
Identifier: /END_POINTS

- where ###  ID number of entity
        PNT ==> POINT
    X-COORD ==> real value of x coordinate (cartesia
    Y-COORD ==> real value of y coordinate (cartesian)
    Z-COORD ==> real value of z coordinate (cartesian)
```

13. Unit Vector Dictionary -

The Unit Vector dictionary contains all of the UNIT VECTOR definitions associated with the part's geomety. Each UNIT VECTOR definition is characterized by three unit vector direction COMPONENTS.

Unit Vector Dictionary Syntax:

```
Identifier:  /UNIT_VECTORS
Attribute:     VEC### ; X-COMP, Y-COMP, Z-COMP .
Identifier:  /END_UNIT VECTORS

- where ### ==> ID number of entity
        VEC ==> UNIT VECTOR
     X-COMP ==> X COMPONENT of UNIT VECTOR (normalized)
     Y-COMP ==> Y COMPONENT of UNIT VECTOR (normalized)
     Z-COMP ==> Z COMPONENT of UNIT VECTOR (normalized)
```

Each part and part blank design which is to be analysed by the FEATURE-ID system must first be stored in a database text file using the dictionary structure outlined above. Input database examples using this structure appear in Appendix A and B. These databases are for the part and part blank appearing in Figure 5.3A and Figure 5.3B.

5.5.2 Input Database Interpretation:

The input database file is interpreted by the FEATURE-ID system through a lexical scanning process. Each line in the database is read by the program as a character string. The string is then scanned according to its dictionary identity, attributes, and syntax punctuation. Data is then extracted from the string and placed in the system RAM using Pascal record structures.

The structure of the BREP text file database creates a convenient interface capability between the text storage file and the system RAM. The dictionary format supports a considerable amount of data sharing. For instance, points that represent the location of vertices are also used to

define curves and surfaces [13,14]. By organizing the topological and geometric data associated with a part or part blank design into this sharing format, a powerful hierarchical database is created [Figure 5.7]. The FEATURE_ID program takes full advantage of this database structure by directly reading the text file into program variable positions. Once the part and part blank input files are interpreted and placed in memory, subsequent program algorithms extract the machinable feature definitions to be removed from the part blank in order to produce the part.

5.5.3 Input Database Preparation:

The part and part blank design input databases are prepared manually in this research. In order to accomplish this task, it is necessary to analyse a part design drawing and physically build the database according to the format outlined above. Ultimately, a CAD/CAE design system would directly provide part design databases using the BREP dictionary structure as output. CAD/CAE systems are being developed which will handle this task.

5.6 FEATURE-ID OUTPUT DATABASE FORMAT

As discussed in section 3.3, a feature based PDD exchange database could be effectively utilized as input to

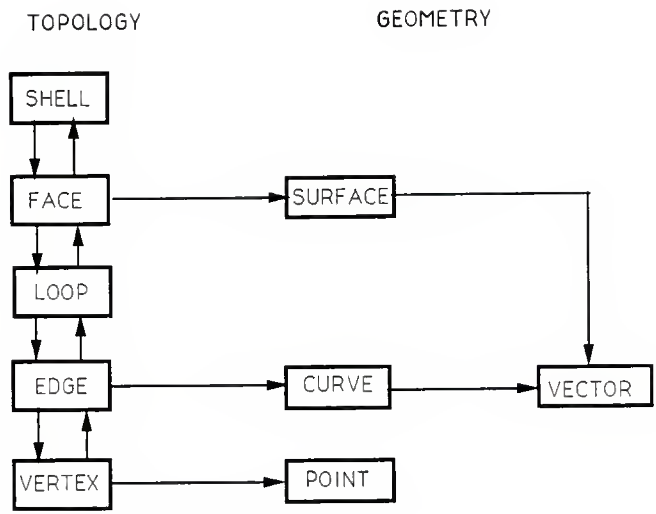


FIGURE 5.7 - DATA SHARING HIERARCHICAL STRUCTURE OF INPUT DATABASE.

CAM systems. Of primary interest in this research is developing an exchange database which organizes part manufacturing data for input to CAPP and NC program generator CAM systems. The FEATURE-ID CAD-to-CAM data translation system produces this type of exchange database as its final output.

5.6.1 Exchange Database Structure:

The structure of the FEATURE-ID output exchange database is designed to provide information to CAM systems in terms of the features to be machined from a given part blank in order to produce a given part. This is accomplished by first analysing the part and part blank input design databases. The function of this analysis is to identify features to be removed from the part blank. Once a feature is identified, it is stored within the exchange database structure. Each feature defined within the exchange database is then furthur broken down into individual components. Pertinent topological and geometric data from the part and part blank design databases is then associated with each feature component and placed into the exchange database. The resulting exchange database is essentially an organized collection of individual BREP databases, each describing a component of a feature to be removed from the part blank.

5.6.2 Output Database Dictionary Structure:

The dictionary structure of the output exchange database is similar to that of the input design database dictionary structure documented in Section 5.5. However, the Part dictionary found in the part design database is replaced by a Feature-Removal-Model dictionary. Also, two additional dictionaries are added in order to allow for feature organization within the exchange database. These are the Feature dictionaries, and the Feature Component dictionaries [Figure 5.8]. The structure of the output database is outlined below.

1. Feature Removal Model dictionary -

The level-one Feature Removal Model dictionary encompasses the entire feature-based exchange database output by the FEATURE-ID system. Contained within this dictionary are the level-two Header dictionary and any number of level-two Feature dictionaries.

Feature Removal Model Dictionary Syntax:

```
     Identifier:  /FEATURE_REMOVAL_MODEL
                  <Header Dictionary>
                  <Feature Dictionary>
                  <Feature Dictionary>
                     .
                     .
                     .
                  <Feature Dictionary>
     Identifier:  /END_FEATURE_REMOVAL_MODEL
```

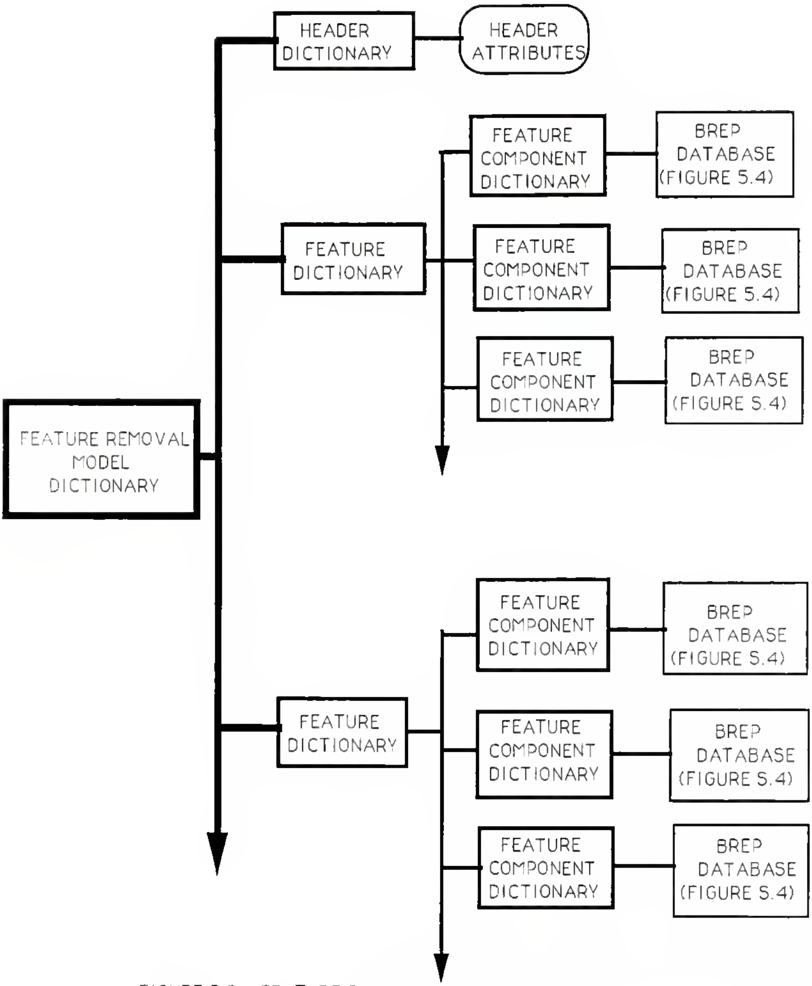


FIGURE 5.8 - FEATURE BASED EXCHANGE DATABASE DICTIONARY STRUCTURE.
FEATURE-ID SYSTEM OUTPUT.

2. Feature Dictionaries -

Each Feature Removal Model dictionary contains several level-two Feature dictionaries. Each of these Feature dictionaries contain several Feature Component Dictionaries. One Feature dictionary contains all information needed to describe a particular feature to be removed from the part blank. The FEATURE_ID system has the capability of analyzing "Top_Face" and "Pocket" features of removal [Figure 5.9].

Feature Dictionary Syntax:

For "Top_Face" Feature -

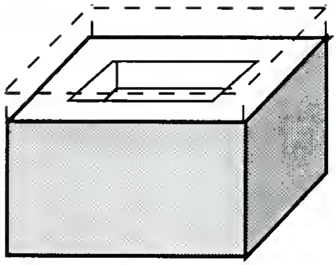
```
     Identifier:  /FEATURE_TOP_FACE
                  <Feature Component Dictionary>
                  <Feature Component Dictionary>
                     .
                     .
                     .
                  <Feaature Comppponent Dictionary>
     Identifier:  /END_FEATURE_TOP_FACE
```

For "Pocket" Feature -

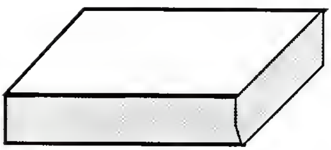
```
     Identifier:  /FEATURE_POCKET
                  <Feature Component Dictionary>
                  <Feature Component DIctionary>
                     .
                     .
                     .
                  <Feature Component Dictionary>
     Identifier:  /END_FEATURE_POCKET
```

3. Feature Component Dictionary -

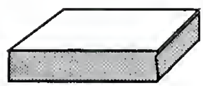
Each level-two Feature dictionary contains several level-three Feature Component dictionaries. Each of these



EXAMPLE PART (TOP-FACE AND POCKET REMOVED)



TOP FACE FEATURE



POCKET FEATURE

FIGURE 5.9 - FEATURES OF REMOVAL.

Feature Component dictionaries contains a BREP database defining a component of a feature in terms of its topology and geometry. The components associated with the "Top_Face" feature are "Entry_Plane" and "Check_Plane". The components associated with the "Pocket" feature are "Side_Plane" and "Bottom_Plane" [Figure 5.10].

Feature Component Dictionary Syntax:

For "Entry_Plane" Feature Component -

```
     Identifier:   /ENTRY_PLANE
                   <BREP data>
     Identifier:   /END_ENTRY_PLANE
```

For "Check_Plane" Feature Component -

```
     Identifier:   /CHECK_PLANE
                   <BREP data>
     Identifier:   /END_CHECK_PLANE
```

For "Side_Plane" Feature Component -

```
     Identifier:   /SIDE_PLANE
                   <BREP data>
                   /END_SIDE_PLANE
```

For "Bottom_Plane" Feature Component -

```
     Identifier:   /BOTTOM_PLANE
                   <BREP data>
     Identifier:   /END_BOTTOM_PLANE
```

Each part and part blank feature analysis application submitted to the FEATURE_ID system results in the production of an exchange database using the dictionary structure outlined above. An output exchange database example created

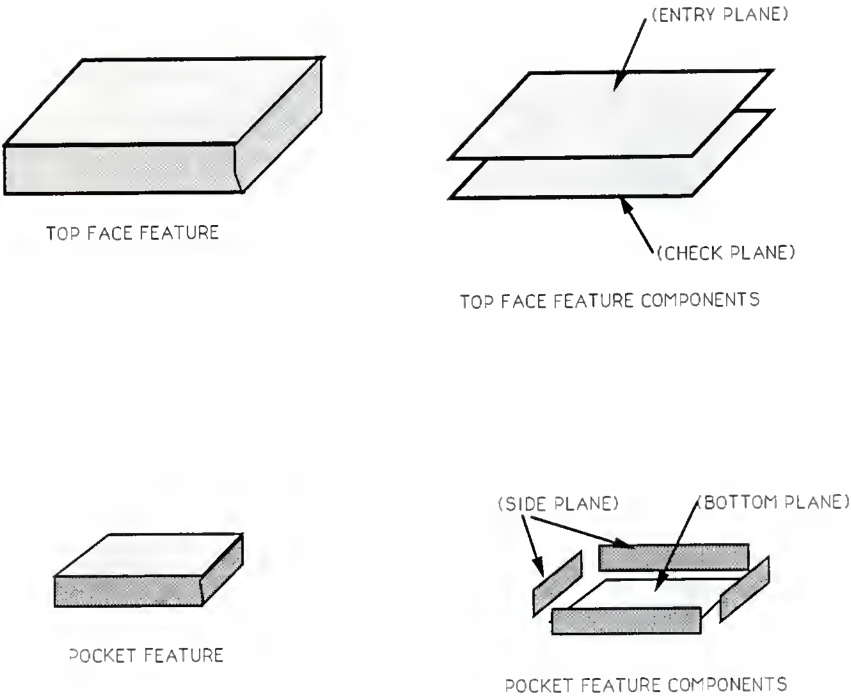


FIGURE 5.10 - FEATURE COMPONENTS.

by the FEATURE-ID system using this structure appears in appendix C. This database is a result of the analysis of the hypothetical part and part blank application appearing in Figure 5.3A and 5.3B.

5.7 FEATURE-ID PROGRAM DOCUMENTATION:

The FEATURE_ID system is an interactive Pascal program which has the capability of reading a BREP part design database and a BREP part blank database formatted using the structure outlined in Section 5.5. In addition, the system can perform a feature of removal extraction analysis which defines top-face and pocket features to be removed from a part blank by an NC machine. The final output of the system is a manufacturing oriented exchange database which contains the data associated with each feature identified during the analysis. The format of this output exchange database is documented in Section 5.6.

This section of the paper presents documentation of the FEATURE-ID program. First, a program walkthrough is presented in reference to the hypothetical part application shown in figure 5.3A and 5.3B. This walkthrough details how to use the FEATURE-ID program in terms of input database preparation and program utilization. Second, documentation of the actual program code is presented. This documentation details the FEATURE-ID program structure in terms of the

individual procedures and functions used in constucting the program.

5.7.1 FEATURE-ID Program Walkthrough:

Part & Part Blank Design Database Input -

As discussed in Section 5.5.3, the part and part blank input databases are prepared manually according to the format documented in Section 5.5. The database representing the part illustrated in Figure 5.3A is contained in appendix A. The database representing the part blank illustrated in figure 5.3B is contained in Appendix B. These databases are built using a text file editor such as Wordstar or the Turbo Pascal editor. Each design database file must have a standard DOS file name and file type. The filename can be any legal string up to 8 characters. The filetype must be "BRP" representing a BREP database. The file names given to the hypothetical part and part blank used in this walkthrough are "PART1.BRP" and "BLANK1.BRP" respectively.

In the IBM PC/XT used in this project, the Turbo Pascal software as well as all FEATURE-ID program code files are stored on a directory called "TP". The name of this resident directory of the Turbo Pascal 4.0 software is not important, and is a function of individual preference. However, two subdirectories of the resident directory must be created to house the input and output files associated

with the FEATURE-ID system operation. The "SOURCE" subdirectory is where the part and part blank input design database files must be located prior to running the FEATURE-ID program. Feature based exchange databse ouputs from the program are stored in the "RESULT" subdirectory. Be sure the directory structure is set up correctly before running the program.

Loading the FEATURE-ID Program -

The FEATURE-ID program code is written using a "unit" structure available with the Turbo Pascal 4.0 compiler. A unit is a collection of constants, data types, variables, procedures, and functions. Each unit is like a separate Pascal program: it can have a main body that is called before the program and does whatever initialization is necessary. In short, a unit is a library of declarations that can be pulled into a main program. This library structure allows the main program to be split up and separately compiled. Limitations on code segment size on the development IBM PC/XT with an 8088 CPU are 64K. The FEATURE-ID program requires 120K total to operate. The Turbo Pascal unit structure handles this by making each unit a separate code segment. Thus the code segment size limit is increased to the size of the machine CPU, 640k in this project, by using the unit structure [12]. Before loading

the FEATURE-ID program, it is important to make sure that all units used in the program are available on the same directory as the Turbo Pascal software. Units needed are as follows:

- Turbo Pascal Utility Units:

1. DOS.TPU	2. CRT.TPU	3. GRAPH.TPU
4. GDRIVER.TPU	5. PRINTER.TPU	6. GKERNEL.TPU
7. GWINDOW.TPU		

- FEATURE-ID Specific Units:

1. PROG.TPU	2. COMMON.TPU	3. PLOT.TPU
4. MODEL.TPU	5. SORT.TPU	6. ARRAYS.TPU
7. READ.TPU	8. TOP_FACE.TPU	9. POCKET.TPU

In order to compile the program, first invoke Turbo Pascal 4.0 by typing the "TURBO" command from the resident directory. Next, load the FEATURE.PAS file into the system. Once the main program is loaded, choose COMPILE with a BUILD option. When compilation is complete, a ready prompt will appear. Choose RUN to invoke the program.

It is recomended that any users of this program should first consult the Turbo Pascal 4.0 manual for basic understanding of the system operation [12].

FEATURE_ID Program Main Menu -

After the FEATURE-ID program has been compiled and is

running, the main program menu appears. From the main menu, the user has the ability to select a part and part blank for feature of removal analysis, model the part and part blank graphically, run the analysis, model the results of the analysis, and print the input or output databases used or produced by the program. Detailed explanations of each main menu option and how they are used in reference to the part walkthrough are presented below:

- Source File Operations:

STORE Command:

The STORE command is used to initialize a newly developed part or part blank file. Each input design database file must be initialized by using the STORE option before subsequent program functions can be run. Once the STORE procedure is completed for a given part or part blank, it need not be repeated as the initialized database format is stored on the hard drive.

The function of the STORE procedure is to break up the complete BREP database file into several pieces. Each piece contains all data associated with one topological or geometric dictionary within the design database file. This information is then stored in separate files. By doing this, the efficiency of the lexical scanning of the database is improved. Since not all program procedures require all

of the data associated with a particular part design, scans can be made selectively.

To use the STORE function, type STORE (in capitals) at the main menu command prompt. Once prompted, enter the filename of the part or part blank to initialize. To initialize "PART1.BRP", enter PART1 and hit return. As each dictionary is initialized, updates will appear on the screen. Upon completion of the initialization, control will return to the main menu.

- Selection for Display :

SELECT Command:

The SELECT command is used to choose an initialized part or part blank for viewing. A part must first be identified using the SELECT command before the MODEL command will function. To select a part or part blank, enter SELECT at the main menu command prompt. To select "PART1", type PART1 in response to the SELECT prompt and hit return. Once a part is selected, a status line at the top of the main menu will be updated with the selected part name. The selected part will remain active until another is chosen.

VIEW Command:

The VIEW command is used to set viewing options which tailor the graphic image display of the selected part. The

VIEW procedure must be run in addition to the SELECT command before MODEL or D_FEAT graphic routines are invoked. To set the view options for the graphic display, type VIEW at the main menu prompt. Once the VIEW CONTROL menu appears, four view options are presented. Choose one of these options, "F" for front view, "L" for left view, "O" for orthogonal view, or "U" for a user specified view to set the view. If a user specified view is requested, a sub-menu will appear. X, Y, and Z axis rotation angles must then be entered to set the view. Once the view options are set, a prompt will appear which allows for optional display of point coordinates in the view. Next, the coordinate origin or "home" point must be set according to a screen pixel location. Pixel resolution is 0 to 639 left to right, and 0 to 319 top to bottom. The final parameter requested to set the view is a scale factor. Each 100-by-100 pixels represents 1 square inch using a 1-to-1 scale factor. By increasing the scale factor to 2, each 100-by-100 pixels represents 4 square inches. Depending on the actual size of the part to be displayed, scale factor can be altered to fit the part picture on to the screen. After setting all view parameters, program control is returned to the main menu.

For the program walkthrough application, the view of "PART1" is set by first choosing "O" for an orthogonal view. Next, inclusion of point coordinates is selected. Also, the

home point is set at pixel 100 in the X direction, and 150 in the Z direction. Finally, the scale factor is set to 2. Once the view is set, any graphical modeling procedures will utilize the requested viewing parameters. After returning to the main menu, a view status line will appear at the top of the screen showing a summary of requested view options.

- DIsplay:

MODEL Command:

The MODEL command is used to view a part or part blank which has been selected. The FEATURE-ID program will switch to graphics mode and a wire frame model of the part requested will be drawn on the screen. The part image will be drawn in accordance to the current view settings chosen in the VIEW routine. After inspection of the image, hitting return brings the program control back to the main menu.

For the program walkthrough application, "PART1" is modeled by entering MODEL at the main menu prompt. Due to the intensive disk access requirements inherent in the FEATURE_ID program, the time taken to produce the part drawing is noticably long.

- Analysis:

RUN Command:

The RUN command is used to invoke a feature removal

analysis. Once the RUN procedure is invoked, three filename identifiers are requested. First, the part name refering to the desired part input database filename is entered. Next, the blank name refering to the desired blank input database filename is entered. Finally, a filename must be specified which identifies a file to be used to store the output exchange database resulting from the analysis. After entering the RUN analysis file parameters, a status window will display the files chosen and the feature removal analysis will begin.

As the analysis proceeds, program operational status lines will appear on the screen indicating the analysis results as they occur. After each feature is completely analysed, the program will pause to allow for inspection. Once ready to continue, the operator must press return to continue the analysis. Once the analysis is complete, individual feature results are combined into a completed feature based manufacturing exchange database and stored in the "RESULT" subdirectory under the file name requested and a filetype of "FID". This is the final output of the FEATURE-ID system.

For the program walkthrough application, a feature removal analysis was RUN for "PART1" subject to "BLANK1". The requested analysis storage name was "ANAL_001". The first analysis algorithm in the FEATURE-ID program

identifies the top-face feature. As this feature is extracted, status lines appear on the screen identifying results as each component of the top_face feature is identified. Similarly, an algorithm is invoked to identify the pocket feature to be removed. Upon completion of the analysis, the exchange database is built and stored in the "RESULT" subdirectory as "ANAL_001.FID" and program control is restored to the main menu.

Documentation of the feature removal algorithms appears in Appendix N and Appendix O.

D_FEAT Command:

The D_FEAT command is used to generate a graphical representation of a feature removal analysis generated using the RUN command. This procedure uses the actual exchange database produced by the FEATURE_ID system feature removal analysis as input to its graphic routines. After entering the D_FEAT procedure, the file name of an exchange database desired for modeling is requested. Once the request is entered, the program enters into graphics mode, and an image is produced. The image created illustrates the features of removal found in the RUN analysis.

Initially, a dashed line image of the part blank is displayed on the screen. Next, the planar components of the top-face feature are displayed showing the feature of

removal in reference to the part blank. Similarly, the planar components of the pocket features found are displayed. Top_face features are shown in green, and pocket features are shown in cyan.

For the program walkthrough application, the "ANAL_001" exchange database produced by the RUN procedure was chosen for display. The resulting display showed the image of "BLANK1" initially. In addition the top_face and pocket feature were drawn with reference to the "BLANK1" image. The view orientation used by the D_FEAT procedure is a result of the VIEW selection procedure. After visual analysis of the feature removal display was completed, a carriage return returns control to the main menu.

PRINT Command:

The PRINT command is used to generate hard copies of input and output databases resident on the "SOURCE" or "RESULT" subdirectories. Input design databases as well as output exchange databases can be printed using this command. Once the PRINT procedure is invoked, three request types appear. Choosing "A" initiates the print of an input design database. Choosing a "B" initiates the print of an output exchange database. Choosing "C" aborts the PRINT procedure. After choosing an option, an appropriate submenu will appear requesting a desired filename to print. The actual printout

provided will contain a header identifiying the requested print file name and directory. Control will return to the main menu before the print is completed.

- Program

EXIT Commmand:

The exit command can be invoked at any time from the main menu. This command will terminate program execution and return the user to the Turbo Pascal window.

5.7.2 FEATURE-ID Program Code Documentation:

The FEATURE_ID program is written using a traditional hierarchical structure. By using the Turbo Pascal unit structures described in Section 5.7.1, it was possible to design the program as a single top-down entity in spite of its size. Although individual procedures and functions reside in different units, they are pulled into the main program as needed.

Unit Documentation -

The FEATURE_ID program uses nine custom written units as well as three Turbo Pascal package units in its code structure [Table 5.1a, 5.1b]. Each unit contains a collection of procedures, functions, and variable declarations used in the program. Descriptions of the content of each unit will be presented below. Actual code will be contained in Appendices X through Y and is

PROGRAM OR UNIT	TURBO PASCAL UNITS INCLUDED.	FEATURE_ID UNITS INCLUDED	TURBO ROUTINE CODE	CUSTOM ROUTINE CODE
Program FEATURE_ID	CRT	COMMON PROG		FEATURE_ID (main program)
Unit PROG	CRT GRAPH PRINTER	COMMON SORT PLOT MODEL READ TOP_FACE POCKET		STORE SELECT SET_VIEW MODEL_IT RUN D_FEAT PRINT
Unit COMMON				DUMMY
Unit SORT	CRT	COMMON		SORT_PART SORT_PART_2 SPLIT
Unit PLOT		COMMON		CONVERT_TO_ ORTH_REAL CONVERT_TO_ ORTH_INT
Unit MODEL	DOS CRT GRAPH	COMMON		MOD_PLOT_POINTS MOD_DRAW_LINES DRAW_FRAME DRAW_FEATURE_ FRAME DRAW_FEATURE_ TRACK
Unit READ		COMMON ARRAYS		READ_DATA
Unit TOP_FACE	CRT	COMMON READ		FIND_TOP_FACES STORE_TFC_DATA

TABLE 5 1A - FEATURE_ID PROGRAM AND UNIT CONTENTS

PROGRAM OR UNIT	TURBO PASCAL UNITS INCLUDED.	FEATURE_ID UNITS INCLUDED	TURBO ROUTINE CODE	CUSTOM ROUTINE CODE
Unit POCKET	CRT	COMMON READ		FIND_POCKETS CONSOLIDATE STORE_POC_DATA
Unit ARRAYS	CRT	COMMON		READ_VECTORS READ_POINTS READ_CURVES READ_SURFACES READ_VERTICES READ_EDGES READ_LOOPS READ_FACES READ_SHELLS
Unit CRT			TEXTMODE TEXTCOLOR CLRSCR	
Unit GRAPH			INITGRRAPH CLOSEGRAPH PUTPIXEL SETCOLOR SETLINESTYLE SETTEXTSTYLE STR LINE OUTTEXTXY SETTEXT- JUSTIFY	

TABLE 5.1B - FEATURE_ID PROGRAM AND UNIT CONTENTS CONTINUED

referred to in the descriptions.

Unit PROG:

Appendix E

The unit PROG contains the second level procedures of the FEATURE_ID program. Each of these 7 procedures represents one module of the FEATURE_ID program. These procedures are called by the main program upon request of the user and execute in a modular fashion. Each operation available to the user contained in the main menu is associated with one of the procedures contained in this unit. Once invoked, these procedures control the operation of a particular program module independent of all other program modules. When all program functions associated with a level-two procedure module are finished, program control is returned to the main program routine.

Unit COMMON:

Appendix F

The unit Common is used as a global variable declaration common. The Turbo Pascal unit structure allows for both local variable declaration within a unit as well as global variable declaration available to the unit and any other programs or units using the unit. This allows for convenient centralization of variable declarations for most

of the variables in the FEATURE_ID program. By including the Common unit in the main program and other program units, global variables are defined throughout the program. The Dummy procedure is included to fulfill the unit syntax requirements, and actually does nothing.

Unit SORT:

Appendix G

Unit SORT contains the level-three and level-four procedures used by the STORE procedure. The function of these routines is to initialize a BREP input file by breaking it up into more managable pieces. Each piece represents one dictionary associated with a part database file. By providing this initialization, the program can more easily access input file data as it is needed.

Unit PLOT:

Appendix H

The PLOT unit contains two level-three procedures used by both the MODEL_IT and D_FEAT procedures. These routines calculate the projections necessary to provide a three dimensional image of a part model for graphic display.

Unit MODEL:

Appendix I

The MODEL unit contains five level-three procedures

used by both the MODEL_IT and D_FEAT procedures. These procedures are used to create a graphical image of a part for a modeling request by the user.

Unit READ:

Appendix J

Unit READ contains one level-three procedure used by the level-two D_FEAT and MODEL_IT procedures as well as the level-three FIND_TOP_FACES and FIND_POCKETS procedures. This routine coordinates the loading of text file data into the program variables in system RAM.

Unit TOP_FACE:

Appendix K

The TOP_FACE unit contains the procedures used in the top face feature removal analysis. One level-three routine and one level-four routine are contained in this unit and are used by the level-two RUN procedure in performing this function.

Unit POCKET:

Appendix L

The POCKET unit contains the procedures used in the pocket feature removal analysis. Two level-four routines as well as one level-three routine are contained in this unit and are used by the level-two RUN procedure in performing

this function.

Unit ARRAYS:

Appendix M

The ARRAYS unit contains the procedures used for lexical scanning of input files. The function of these units is to scan for data contained in the text based input files and load this information into program variable locations in RAM. Nine level-four procedures are contained in the ARRAYS unit. These procedures are coordinated by the READ_DATA routine found in unit READ in order to perform this function.

Turbo Pascal Units:

The units CRT, GRAPH, and PRINTER are Turbo Pascal standard package units. The CRT unit contains routines which control the IBM PC/XT features such as screen mode control, colors, windows, and sound. The GRAPH unit contains routines needed to implement graphics calls from the program. The PRINTER unit provides routines which enable easy interface of the PC and printer through the program. The FEATURE_ID program and most of its units use various functions contained in the above units. Individual details for a particular standard unit procedure can be found in the Turbo Pascal 4.0 manual [12].

FEATURE_ID Program code documentation -

The FEATURE_ID program is designed with a traditional tree structure. Once running, the program remains active throughout the entire analysis process. Individual modules branch from the main program to perform specific functions as requested by the user.

Documentation of the program will be organized in parallel with the design structure. Each of the level-two program module procedures used in the program will be described briefly below. Details of data transfer, function and structure as well as reference to actual code will be included in each routine description.

Program FEATURE_ID:

Appendix D

Unit: none

Level: one

Parent: none

Children:

- Custom Routines:

    Proc. STORE, Proc. SELECT, Proc. SET_VIEW,
    Proc. MODEL_IT, Proc. RUN, Proc. D_FEAT,
    Proc. PRINT,

- Turbo System Routines:

    Proc. CLRSCR, Proc. TEXTMODE, Proc. TEXTCOLOR

Description:

The level-one FEATURE_ID routine is the main program in the FEATURE_ID code [Figure 5.11a]. This rouutine provides an interface with the user by first writing a main menu on to the CRT. At this point, any valid commands entered are picked up by the routine, and program control is directed to the appropriate level-two procedure. After the level-two procedure has completed execution, program control returns to the FEATURE_ID main program and the next command can be entered.

Procedure STORE:

Appendix E

Unit: PROG

Level: Two

Parent: FEATURE_ID

Children:

- Custom Routines:

  Proc. SORT_PART, Proc. SPLIT

- Turbo System Routines:

  Proc. TEXTCOLOR, Proc. CLRSCR,

Description:

The level-two STORE procedure is one of the modules

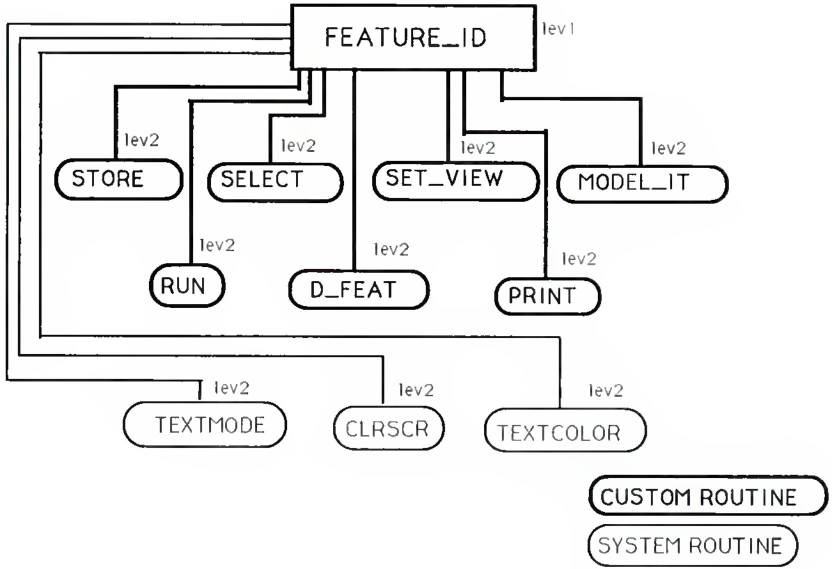


FIGURE 5.11A - FEATURE_ID Program Schematic
Main program, level two program modules.

called by the FEATURE_ID main program [Figure 5.11b]. This module is invoked by entering the "STORE" command from the main menu. The function of the STORE module is to initialize a part model file by splitting it up into smaller files prior to analysis. Each of these smaller files contains the information specific to one part feature dictionary type as described in Section 5.5. As the FEATURE_ID program runs a part feature removal analysis, it references the files created by the STORE procedure as described in Section 5.7.1.

Procedure SELECT:

Appendix: E

Unit: PROG

Level: Two

Parent: FEATURE_ID

Children:

- Turbo System Routines:

  Proc. CLRSCR, Proc. TEXTCOLOR

Description:

The level-two SELECT procedure is a simple program module called by the FEATURE_ID main program upon entering

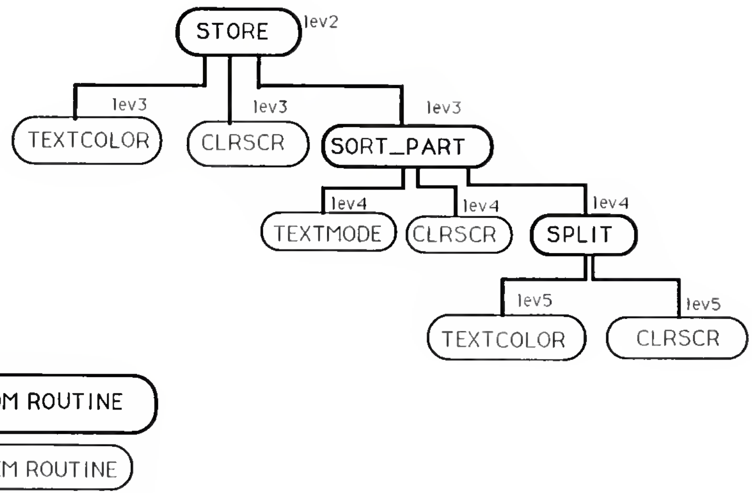


FIGURE 5.11B - Schematic of STORE program module

the "SELECT" command from the main menu [Figure 5.11c]. This module directs the user in selecting a part model for analysis and initializes the needed program variables according to user response.

Procedure SET_VIEW:

Appendix E

Unit: PROG

Level: Two

Parent: FEATURE_ID

Children:

- Turbo System Routines:

  Proc. CLRSCR, Proc. TEXTCOLOR

Description:

The level-two SET_VIEW procedure is also a simple program module called to by the main program [Figure 5.11d]. This module directs the user in setting view variables which are used in graphical image production by other program procedures. The module is invoked by the "view" command from the main menu.

Procedure MODEL_IT:

Appendix: E

Unit: PROG

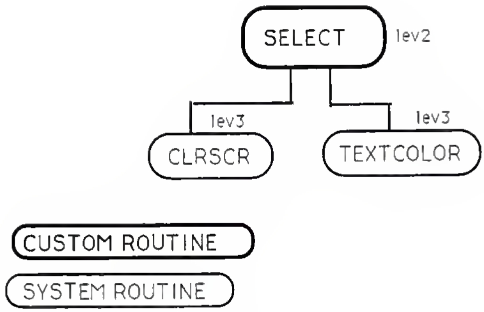


FIGURE 5.11C - Schematic of SELECT program module.

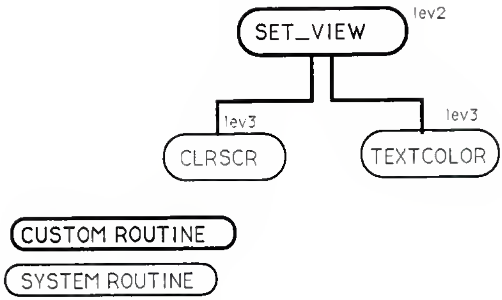


FIGURE 5 1 1D - Schematic of SET_VIEW program module.

Level: Two

Parent: FEATURE_ID

Children:

- Custom Routines:

  Proc. READ_DATA, Proc. CONVERT_TO_ORTH_INT,

  Proc. DRAW_FRAME, Proc. CONVERT_TO_ORTH_REAL,

  Proc. MOD_PLOT_POINTS, Proc. READ_SHELLS,

  Proc. READ_FACES, Proc. READ_LOOPS,

  Proc. READ_EDGES, Proc. READ_VERTICES,

  Proc. READ_VECTORS, Proc. READ_POINTS,

  Proc. READ_CURVES, Proc. READ_SURFACES

- Turbo System Routines:

  Func. VAL, Proc. SETCOLOR, Proc. SETLINESTYLE,

  Proc. LINE, Proc. SETTEXTSTYLE, Proc. SETTEXTJUSTIFY,

  Proc. OUTTEXTXY, Proc. PUTPIXEL, Proc. STR,

  Proc. INITGRAPH, Proc. CLOSEGRAPH

Description:

The level-two MODEL_IT procedure program module is called by the main program by the "model" command [Figure 5.11e]. This module is used to display a graphical image of a part or part blank which has been selected for analysis. Data is read into the program variables using the READ_DATA procedure. The CONVERT_TO_ORTH routines then convert the three dimensional geometric part model data into a two

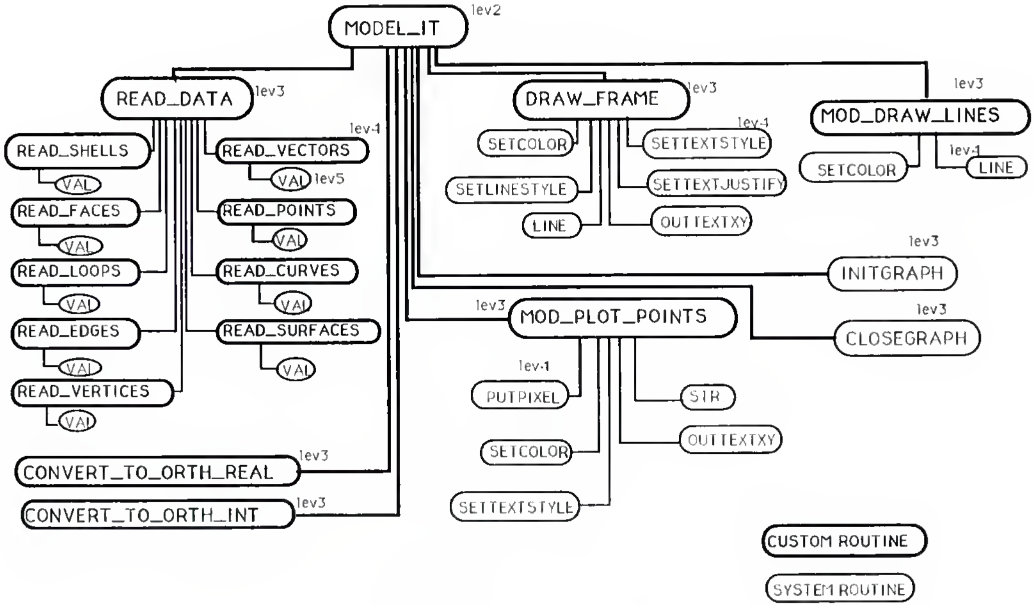


FIGURE 5.11E Schematic of procedure MODEL_IT program module.

dimensional orthogonal representation accordng to the desired view set using the SET_VIEW module. This representation is then stored and used by the MOD_PLOT_POINTS, and MOD_DRAW_LINES procedures to produce the graphical part image on the screen.

Procedure RUN:

Appendix: E

Unit: PROG

Level: Two

Parent: FEATURE_ID

Children:

- Custom Routines:

  Proc. FIND_TOP_FACES, Proc. FIND_POCKETS,
  Proc. READ_DATA, Proc. STORE_TFC_DATA,
  Proc. CONSOLIDATE, Proc. STORE_POC_DATA,
  Proc. READ_SHELLS, Proc. READ_FACES,
  Proc. READ_LOOPS, Proc. READ_EDGES,
  Proc. READ_VERTICES, Proc. READ_VECTORS,
  Proc. READ_POINTS, Proc. READ_CURVES,
  Proc. READ_SURFACES

- Turbo System Routines:

  Func. VAL, Proc. TEXTCOLOR

Description:

The level-two RUN procedure is the central program module of the FEATURE_ID program [Figure 5.11f]. This

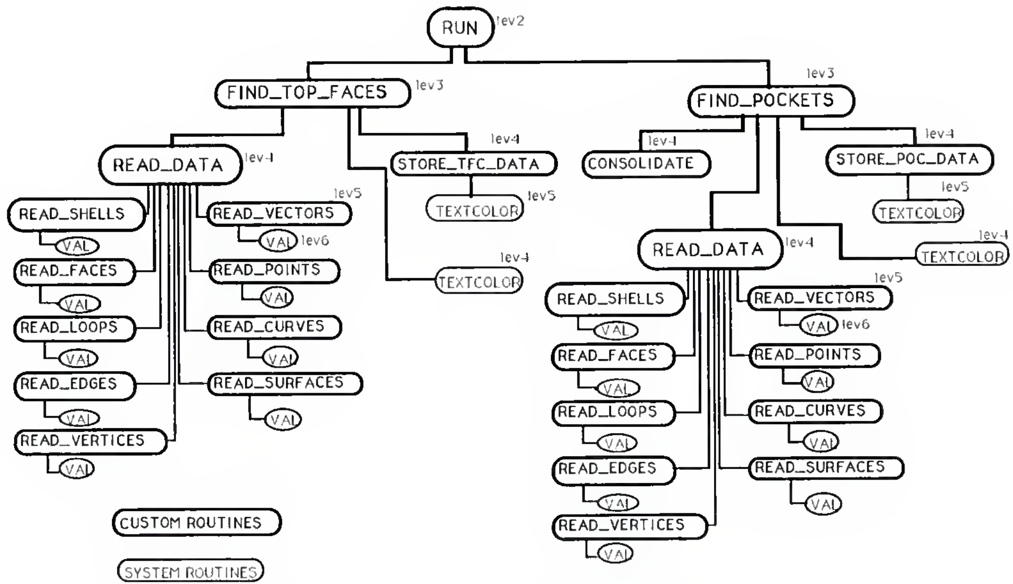


FIGURE 5.11F Schematic of procedure RUN program module

procedure is invoked by entering the "run" command on the main menu. The function of the RUN procedure is to perform the feature removal analysis of a part and part blank model selected by the user.

Once invoked, the RUN module begins execution by directing the user to select the part and part blank to be analysed. After this initialization is completed, the analysis is run in two sections. First, the top-face feature analysis is run. Second, the pockets feature analysis is completed. These are the two types of features which this program is capable of handling.

In the top-face analysis, the FIND_TOP_FACES procedure first finds the entry plane of the top-face feature of removal. This is done by reading in the blank part file, calculating the location of the face or faces in the most positive z plane orientation, and storing their array locations in memory. Next, the program scans the part blank file data and stores all feature dictionary data associated with the entry plane face in the top-face analysis result file using the STORE_TFC_DATA routine. Similar analysis and storage is performed to determine the top-face check plane. Upon completion of the top-face feature removal analysis, resulting data is added to the overall feature removal analysis file specified by the user [Appendix N].

After all functions associated with the top-face

analysis are completed, the pocket feature of removal analysis is invoked. The operation of the FIND_POCKETS procedure is similar in structure to that of the FIND_TOP_FACES routine. The major difference is in the analysis associated with determining a pocket location and storing its data.

The FIND_POCKETS routine locates a pocket by searching for a specific occurance in a part model. This algorithm first locates a compound face on the part surface. This occurs when one or more loop structures on a particular face are surrounded by another loop. Upon finding a compound face, the algorithm analyzes each internal loop to determine if a pocket feature is associated with it. The analysis proceeds by identifying all side faces and the bottom face associated with the pocket. All data associated with each pocket face is then identified and grouped by the CONSOLIDATE procedure. This information is then stored in the result file of the analysis along with the top-face information in the form of a feature based exchange database file [Appendix O].

Procedure D_FEAT:

Appendix E

Unit: PROG

Level: Two

Parent: FEATURE_ID

Children:

- Custom routines:

  Proc. READ_DATA, Proc. CONVERT_TO_ORTH_INT,
  Proc. DRAW_FEATURE_FRAME,
  Proc. DRAW_FEATURE_TRACK,
  Proc. CONVERT_TO_ORTH_REAL, Proc. MOD_PLOT_POINTS,
  Proc. READ_SHELLS, Proc. READ_FACES,
  Proc. READ_LOOPS, Proc. READ_EDGES,
  Proc. READ_VERTICES, Proc. READ_VECTORS,
  Proc. READ_POINTS, Proc. READ_CURVES,
  Proc. READ_SURFACES

- Turbo system routines:

  Func. VAL, Proc. SETCOLOR, Proc. SETLINESTYLE,
  Proc. LINE, Proc. SETTEXTSTYLE,
  Proc. SETTEXTJUSTIFY, Proc. OUTTEXTXY,
  Proc. PUTPIXEL, Proc. STR, Proc. INITGRAPH,
  Proc. CLOSEGRAPH

Description:

The level-two D_FEAT procedure program module is called by the main program for graphically displaying a feature based exchange database analysis [Figure 5.11g]. After running an analysis of a particular part and part blank, this module can be used to show the individual features

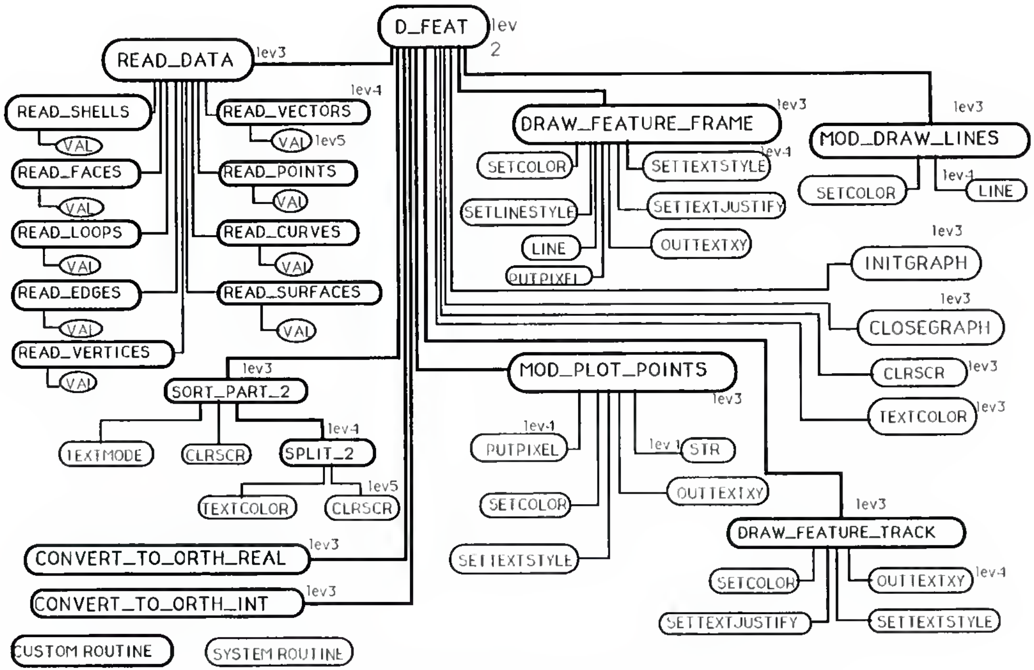


FIGURE 5.11G Schematic of procedure D_FEAT program module.

which define the volumes of material to be removed from the part blank to machine the part.

The D_FEAT routine works much like the MODEL_IT procedure described above. However, it must first sort the exchange database file using the SORT_PART_2 routine before drawing the figure. Since the exchange database is a collection of features, each with a full BREP representation, it takes a significant amount of time to read the data into the program and display the results.

While the results of an analysis is being displayed, each feature will appear color-coded in association with the volume of material feature of removal it is associated with.

Procedure PRINT:

Appendix: E

Unit: PROG

Level: Two

Parent: FEATURE_ID

Children:

- System Routines:

  Proc. CLRSCR, Proc. TEXTCOLOR

Description:

The level-two PRINT procedure program module is invoked from the main program in order to generate hard copies of the various database files used and generated by the FEATURE_ID program [Figure 5.11H].

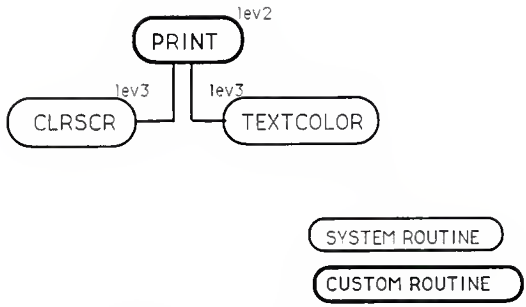


FIGURE 5.11H - Schematic of Procedure PRINT program module.

6. RESULTS, CONCLUSIONS and RECOMMENDATIONS

The objective of the FEATURE_ID project was to develop a computer based system capable of automatically translating a CAD/CAE part design database into a feature based PDD exchange database for use by CAM systems designed to produce NC machine control programs. This section of the paper presents a brief overview of the results of the study. Conclusions and recommendations for future direction in this research area are also discussed.

6.1 FEATURE ID PROJECT RESULTS:

The primary result of this project was the successful development of a very basic prototype computer program system called FEATURE_ID. This system was designed to automatically analyse the BREP database representation of part and part blank models. The end product of a system analysis is the identification of the volumes of material which must be removed from the given part blank in order to generate the desired part. These results are stored in the form of a feature based database which contains the geometric PDD information defining the volumes of removal identified during the analysis.

The FEATURE_ID system is capable of identifying top-face and pocket type volumes of removal during analysis. In addition, the system provides user interfaces which allow

for basic graphic modeling of part input databases and the resulting feature based output databases.

Specific developments were achieved in order to successfully complete the initial FEATURE_ID prototype system. The results of these developments are presented below.

1. Program code was developed to interpret the text based BREP database format used to store part and part blank models for system input.

2. Program code was developed to analyse a given part and part blank model simultaniuosly and automatically determine top face and pocket volumes of removal.

3. A feature based BREP database structure was developed in order to store the results of a FEATURE_ID analysis based on the volume of removal features identified.

6.2 PROJECT CONCLUSIONS:

The scope of the FEATURE_ID project included detailed analysis of the BREP part model input database structure, the development of a program to interpret and analyse these models, and the generation of a feature based BREP model to store the output of the system. The scope of the entire CAD/CAM issue covers far more issues than were addressed in this research. Therfore, I can not make any bold conclusions which support or oppose CAD/CAM development.

However, I can provide conclusions covering the aspect of CAD/CAM research covered in this project.

1. The BREP format used to define databases works well for storing part or part blank models. Models stored in this standard format are easily interpreted by the computer and allow for actual part geometric and manufacturing data definition.

2. Automatic interpretation of part and part blank models for identifying material removal volumes to be fed to NC program generators is a feasable task.

3. Production of feature based BREP databases based on material removal volumes can be accomplished automatically.

6.3 RECOMMENDATIONS:

The FEATURE_ID project focused on the development of automated translation of CAD/CAE design data to CAM manufacturing data. Recommendations surrounding this research include furthur development of the FEATURE_ID system concept as well other CAD/CAM system components needed to develop a truely automated CAD/CAM system. Recommendations stemming from this project are organized into three categories below.

CAD/CAE Part Model Databases -

1. Improve CAD/CAE part model database output.

The function of the CAD/CAE part model database is to store design information resulting from a CAD/CAE work session. In order to successfully link a CAD/CAE system with an automated translation system like FEATURE_ID, two CAD/CAE system functions must be developed.

First, CAD/CAE programs must be able to produce databases meaningful to a computer such as the BREP database used in this project. Second, the CAD/CAE system must automatically generate this database during a design work session.

2. Improve BREP database capability

The BREP database used in the FEATURE_ID project handled only geometric part model information. Improved database structure which can also handle part manufacturing data must be developed.

FEATURE_ID system upgrades -

1. Expand FEATURE_ID software capability

The FEATURE_ID program resulting from this research can interpret two types of volume-of-removal features. Algorithms for interpreting other types of features should be developed. The ability of the system must be expanded in this way to warrent its practical use.

Improvements on database handling must also be

addressed. Storing the input and output part files as text files is fine, but a relational numeric database system might be more efficient. Also, the program should be able to read all needed data into RAM before analysis in order to reduce disk access time.

2. Upgrade system hardware used to run this type of system.

Using a personal computer for the initial FEATURE_ID research was difficult due to the memory requirements of large part models during analysis. Some of the newer personal computer models with powerful CPUs and large RAM would provide a much better environment for this type of system. Use of minicomputers is also recommended.

Feature Based BREP database development -

1. The output of the FEATURE_ID system was a prototype feature base model structure of the volumes of material determined during an analysis. Actual testing of the practicality and usability of this database format as a manufacturing data Exchange database was not part of this project. It would be challenging to develop interfaces with a CAPP system and finally an NC Program generator to develop an NC program based on the content of this database.

In conclusion, I will say that the CAD/CAM research area

provides a huge challenge to anyone choosing to work in it. Many component developments for use in the CAD/CAM area have been accomplished, but the ability of these components to communicate with each other is almost nonexistent at present. I feel that the development and standardization of communicating and interpreting design and manufacturing data are the bottom line key to automated CAD/CAM system success.

REFERENCES

[1] Brooks S.L., Hummel K.E., Wolf M.L., XCUT: A Rule-Based Expert System for the Automated Process Planning of Machined Parts, United States Department of Energy, Bendix Kansas City Division, June 1987.

[2] Harp, Jim, "CAD/CAM: Back to Basics", Manufacturing Engineering, October 1985, p. 61.

[3] Wilms, R., Computer Aided Process Planning: The Implementation and Evolution of CAPP Systems, Master of Science report, Kansas State University Department of Industrial Engineering, 1987.

[4] Wolfe, Philip M., "Computer-Aided Process Planning is Link Between CAD and CAM", Industrial Engineering, August 1986, p. 72.

[5] Anonymous, EC-APT Technical Manual, Elcam Inc., 1987.

[6] Materials Laboratory, Airforce Wright Aeronautical Laboratories, Product Definition Data Interface: Needs Analysis Document, July 1983.

[7] Kral, Irvin H., Numerical Control Programming in APT, New Jersey: Prentice-Hall, 1986, p. 314.

[8] Amstead B.H., Ostwald P.F., Begeman M.L., Manufacturing Processes, New York: John Wiley and Sons, 1977, p. 450

[9] Rouse, Nancy E., "NC Systems Close CAD/CAM Gap", Machine Design, December 1986, p. 88.

[10] Brody, Herb, "CAD Meets CAM", High Technology, May 1987, p. 12.

[11] Woo, Tony C., Interfacing Solid Modeling to CAD and CAM: Data Structures and Algorithms for Decomposing a Solid, University of Michigan, December 1984.

[12] Anonymous, Turbo Pascal 4.0 Manual, Borland International, 1987.

[13] Tu, Juliana S., Hopp, Theodore H., Part Geometry Data in the AMRF, U. S. Department of Commerce: Automated Manufacturing Research Facility, April 1987.

[14] Hopp, Thedore H., AMRF Database Report Format, U.S Department of Commerce: Automated Manufacturing Research Facility, October 1986.

APPENDIX A (Hypothetical Part Example Input Database)

```
/PART_MODEL
/HEADER
  PART NAME = 'PART1'.
/END_HEADER
/TOPOLOGY
/SHELLS
  SHL001 ; FAC001, FAC002, FAC003, FAC004,
         , FAC005, FAC006, FAC007, FAC008, FAC009, FAC010,
         , FAC011 .
/END_SHELLS
/FACES
  FAC001 ; LOP006, LOP007; SUR006 + .
  FAC002 ; LOP001; SUR005 - .
  FAC003 ; LOP002; SUR001 - .
  FAC004 ; LOP003; SUR002 + .
  FAC005 ; LOP004; SUR003 + .
  FAC006 ; LOP005; SUR004 - .
  FAC007 ; LOP008; SUR007 + .
  FAC008 ; LOP009; SUR008 - .
  FAC009 ; LOP010; SUR009 - .
  FAC010 ; LOP011; SUR010 + .
  FAC011 ; LOP012; SUR011 + .
/END_FACES
/LOOPS
  LOP001 ; EDG001 + , EDG002 + , EDG003 - , EDG004 - .
  LOP002 ; EDG005 + , EDG009 + , EDG006 - , EDG001 - .
  LOP003 ; EDG006 + , EDG010 + , EDG007 - , EDG002 - .
  LOP004 ; EDG008 + , EDG011 + , EDG007 - , EDG003 - .
  LOP005 ; EDG005 + , EDG012 + , EDG008 - , EDG004 - .
  LOP006 ; EDG009 + , EDG010 + , EDG011 - , EDG012 - .
  LOP007 ; EDG021 + , EDG022 + , EDG023 - , EDG024 - .
  LOP008 ; EDG017 + , EDG021 + , EDG018 - , EDG013 - .
  LOP009 ; EDG018 + , EDG022 + , EDG019 - , EDG014 - .
  LOP010 ; EDG020 + , EDG023 + , EDG019 - , EDG015 - .
  LOP011 ; EDG017 + , EDG024 + , EDG020 - , EDG016 - .
  LOP012 ; EDG013 + , EDG014 + , EDG015 - , EDG016 - .
/END_LOOPS
/EDGES
  EDG001 ; VTX001 , VTX002 ; CRV001 + .
  EDG002 ; VTX002 , VTX003 ; CRV002 + .
  EDG003 ; VTX004 , VTX003 ; CRV003 + .
  EDG004 ; VTX001 , VTX004 ; CRV004 + .
  EDG005 ; VTX001 , VTX005 ; CRV005 + .
  EDG006 ; VTX002 , VTX006 ; CRV006 + .
  EDG007 ; VTX003 , VTX007 ; CRV007 + .
```

```
  EDG008 ; VTX004 , VTX008 ; CRV008 + .
  EDG009 ; VTX005 , VTX006 ; CRV009 + .
  EDG010 ; VTX006 , VTX007 ; CRV010 + .
  EDG011 ; VTX008 , VTX007 ; CRV011 + .
  EDG012 ; VTX005 , VTX008 ; CRV012 + .
  EDG013 ; VTX013 , VTX014 ; CRV013 + .
  EDG014 ; VTX014 , VTX015 ; CRV014 + .
  EDG015 ; VTX016 , VTX015 ; CRV015 + .
  EDG016 ; VTX013 , VTX016 ; CRV016 + .
  EDG017 ; VTX013 , VTX009 ; CRV017 + .
  EDG018 ; VTX014 , VTX010 ; CRV018 + .
  EDG019 ; VTX015 , VTX011 ; CRV019 + .
  EDG020 ; VTX016 , VTX012 ; CRV020 + .
  EDG021 ; VTX009 , VTX010 ; CRV021 + .
  EDG022 ; VTX010 , VTX011 ; CRV022 + .
  EDG023 ; VTX012 , VTX011 ; CRV023 + .
  EDG024 ; VTX009 , VTX012 ; CRV024 + .
/END_EDGES
/VERTICES
  VTX001 ; PT001 .
  VTX002 ; PT002 .
  VTX003 ; PT003 .
  VTX004 ; PT004 .
  VTX005 ; PT005 .
  VTX006 ; PT006 .
  VTX007 ; PT007 .
  VTX008 ; PT008 .
  VTX009 ; PT009 .
  VTX010 ; PT010 .
  VTX011 ; PT011 .
  VTX012 ; PT012 .
  VTX013 ; PT013 .
  VTX014 ; PT014 .
  VTX015 ; PT015 .
  VTX016 ; PT016 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETRY
/SURFACES
  SUR001 ; PLANE ; VEC002 ; 0.00000 .
  SUR002 ; PLANE ; VEC001 ; 4.00000 .
  SUR003 ; PLANE ; VEC002 ; 4.00000 .
  SUR004 ; PLANE ; VEC001 ; 0.00000 .
  SUR005 ; PLANE ; VEC003 ; 0.00000 .
  SUR006 ; PLANE ; VEC003 ; 2.00000 .
```

APPENDIX A (Hypothetical Part Example Input Database)

```
  SUR007 ; PLANE ; VEC002 ; 1.00000 .
  SUR008 ; PLANE ; VEC001 ; 3.00000 .
  SUR009 ; PLANE ; VEC002 ; 3.00000 .
  SUR010 ; PLANE ; VEC001 ; 1.00000 .
  SUR011 ; PLANE ; VEC003 ; 1.75000 .
/END_SURFACES
/CURVES
  CRV001 ; LINE ; PT001, VEC001 .
  CRV002 ; LINE ; PT002, VEC002 .
  CRV003 ; LINE ; PT003, VEC001 .
  CRV004 ; LINE ; PT004, VEC002 .
  CRV005 ; LINE ; PT001, VEC003 .
  CRV006 ; LINE ; PT002, VEC003 .
  CRV007 ; LINE ; PT003, VEC003 .
  CRV008 ; LINE ; PT004, VEC003 .
  CRV009 ; LINE ; PT005, VEC001 .
  CRV010 ; LINE ; PT006, VEC002 .
  CRV011 ; LINE ; PT007, VEC001 .
  CRV012 ; LINE ; PT008, VEC002 .
  CRV013 ; LINE ; PT013, VEC001 .
  CRV014 ; LINE ; PT014, VEC002 .
  CRV015 ; LINE ; PT015, VEC001 .
  CRV016 ; LINE ; PT016, VEC002 .
  CRV017 ; LINE ; PT013, VEC003 .
  CRV018 ; LINE ; PT014, VEC003 .
  CRV019 ; LINE ; PT015, VEC003 .
  CRV020 ; LINE ; PT016, VEC003 .
  CRV021 ; LINE ; PT009, VEC001 .
  CRV022 ; LINE ; PT010, VEC002 .
  CRV023 ; LINE ; PT011, VEC001 .
  CRV024 ; LINE ; PT012, VEC002 .
/END_CURVES
/POINTS
  PNT001 ; 0.00000, 0.00000, 0.00000 .
  PNT002 ; 4.00000, 0.00000, 0.00000 .
  PNT003 ; 4.00000, 4.00000, 0.00000 .
  PNT004 ; 0.00000, 4.00000, 0.00000 .
  PNT005 ; 0.00000, 0.00000, 2.00000 .
  PNT006 ; 4.00000, 0.00000, 2.00000 .
  PNT007 ; 4.00000, 4.00000, 2.00000 .
  PNT008 ; 0.00000, 4.00000, 2.00000 .
  PNT009 ; 1.00000, 1.00000, 2.00000 .
  PNT010 ; 3.00000, 1.00000, 2.00000 .
  PNT011 ; 3.00000, 3.00000, 2.00000 .
  PNT012 ; 1.00000, 3.00000, 2.00000 .
```

```
  PNT013 ; 1.00000, 1.00000, 1.75000 .
  PNT014 ; 3.00000, 1.00000, 1.75000 .
  PNT015 ; 3.00000, 3.00000, 1.75000 .
  PNT016 ; 1.00000, 3.00000, 1.75000 .
/END_POINTS
/UNIT_VECTORS
  VEC001 ; 1.00000, 0.00000, 0.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
  VEC003 ; 0.00000, 0.00000, 1.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_PART_MODEL
```

Appendix B (Hypothetical Part Blank Example Input Database)

```
/PART_MODEL
/HEADER
  PART NAME = 'BLANK1'.
/END_HEADER
/TOPOLOGY
/SHELLS
  SHL001 ; FAC001, FAC002, FAC003, FAC004,
         , FAC005, FAC006 .
/END_SHELLS
/FACES
  FAC001 ; LOP001; SUR001 - .
  FAC002 ; LOP002; SUR002 + .
  FAC003 ; LOP003; SUR003 - .
  FAC004 ; LOP004; SUR004 + .
  FAC005 ; LOP005; SUR005 + .
  FAC006 ; LOP006; SUR006 - .
/END_FACES
/LOOPS
  LOP001 ; EDG009 + , EDG010 + , EDG011 - , EDG012 - .
  LOP002 ; EDG001 + , EDG002 + , EDG003 - , EDG004 - .
  LOP003 ; EDG003 + , EDG009 + , EDG006 - , EDG001 - .
  LOP004 ; EDG007 + , EDG010 - , EDG006 - , EDG002 + .
  LOP005 ; EDG008 + , EDG011 + , EDG007 - , EDG003 - .
  LOP006 ; EDG008 + , EDG002 - , EDG003 - , EDG004 + .
/END_LOOPS
/EDGES
  EDG001 ; VTX001 , VTX002 ; CRV001 + .
  EDG002 ; VTX002 , VTX003 ; CRV002 + .
  EDG003 ; VTX004 , VTX003 ; CRV003 + .
  EDG004 ; VTX001 , VTX004 ; CRV004 + .
  EDG005 ; VTX001 , VTX005 ; CRV005 + .
  EDG006 ; VTX002 , VTX006 ; CRV006 + .
  EDG007 ; VTX003 , VTX007 ; CRV007 + .
  EDG008 ; VTX004 , VTX008 ; CRV008 + .
  EDG009 ; VTX005 , VTX006 ; CRV009 + .
  EDG010 ; VTX006 , VTX007 ; CRV010 + .
  EDG011 ; VTX008 , VTX007 ; CRV011 + ..
  EDG012 ; VTX005 , VTX008 ; CRV012 + .
/END_EDGES
/VERTICES
  VTX001 ; PT001 .
  VTX002 ; PT002 .
  VTX003 ; PT003 .
  VTX004 ; PT004 .
  VTX005 ; PT005 .
```

```
  VTX006 ; PT006 .
  VTX007 ; PT007 .
  VTX008 ; PT008 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETRY
/SURFACES
  SUR001 ; PLANE ; VEC003 ; 2.20000 .
  SUR002 ; PLANE ; VEC003 ; 0.00000 .
  SUR003 ; PLANE ; VEC002 ; 0.00000 .
  SUR004 ; PLANE ; VEC001 ; 4.00000 .
  SUR005 ; PLANE ; VEC002 ; 4.00000 .
  SUR006 ; PLANE ; VEC001 ; 0.00000 .
/END_SURFACES
/CURVES
  CRV001 ; LINE ; PT001, VEC001 .
  CRV002 ; LINE ; PT002, VEC002 .
  CRV003 ; LINE ; PT003, VEC001 .
  CRV004 ; LINE ; PT004, VEC002 .
  CRV005 ; LINE ; PT001, VEC003 .
  CRV006 ; LINE ; PT002, VEC003 .
  CRV007 ; LINE ; PT003, VEC003 .
  CRV008 ; LINE ; PT004, VEC003 .
  CRV009 ; LINE ; PT005, VEC001 .
  CRV010 ; LINE ; PT006, VEC002 .
  CRV011 ; LINE ; PT007, VEC001 .
  CRV012 ; LINE ; PT008, VEC002 .
/END_CURVES
/POINTS
  PNT001 ; 0.00000, 0.00000, 0.00000 .
  PNT002 ; 4.00000, 0.00000, 0.00000 .
  PNT003 ; 4.00000, 4.00000, 0.00000 .
  PNT004 ; 0.00000, 4.00000, 0.00000 .
  PNT005 ; 0.00000, 0.00000, 2.20000 .
  PNT006 ; 4.00000, 0.00000, 2.20000 .
  PNT007 ; 4.00000, 4.00000, 2.20000 .
  PNT008 ; 0.00000, 4.00000, 2.20000 .
/END_POINTS
/UNIT_VECTORS
  VEC001 ; 1.00000, 0.00000, 0.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
  VEC003 ; 0.00000, 0.00000, 1.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_PART_MODEL
```

Appendix C (Feature Removal Database of Hypothetical Example)

```
/FEATURE_REMOVAL_MODEL
/HEADER
  PART NAME = PART1
  BLANK NAME = BLANK1
  ANALYSIS NAME = ANAL_001
/END_HEADER
/FEATURE_TOP_FACE
/ENTRY_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC001 ; LOP001; SUR001 - .
/END_FACES
/LOOPS
  LOP001 ; EDG009 + , EDG010 + , EDG011 - , EDG012 - .
/END_LOOPS
/EDGES
  EDG009 ; VTX005 , VTX006 ; CRV009 + .
  EDG010 ; VTX006 , VTX007 ; CRV010 + .
  EDG011 ; VTX008 , VTX007 ; CRV011 + .
  EDG012 ; VTX005 , VTX008 ; CRV012 + .
/END_EDGES
/VERTICES
  VTX005 ; PT005 .
  VTX006 ; PT006 .
  VTX007 ; PT007 .
  VTX008 ; PT008 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR001 ; PLANE ; VEC003 ; 2.20000 .
/END_SURFACES
/CURVES
  CRV009 ; LINE ; PT005, VEC001 .
  CRV010 ; LINE ; PT006, VEC002 .
  CRV011 ; LINE ; PT007, VEC001 .
  CRV012 ; LINE ; PT008, VEC002 .
/END_CURVES
/POINTS
  PNT005 ; 0.00000, 0.00000, 2.20000 .
  PNT006 ; 4.00000, 0.00000, 2.20000 .
  PNT007 ; 4.00000, 4.00000, 2.20000 .
  PNT008 ; 0.00000, 4.00000, 2.20000 .
```

Appendix C (Feature Removal Database of Hypothetical Example)

```
/END_POINTS
/UNIT_VECTORS
  VEC001 ; 1.00000, 0.00000, 0.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
  VEC003 ; 0.00000, 0.00000, 1.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_ENTRY_PLANE
/CHECK_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC001 ; LOP006, LOP007; SUR006 + .
/END_FACES
/LOOPS
  LOP006 ; EDG009 + , EDG010 + , EDG011 - , EDG012 - .
  LOP007 ; EDG021 + , EDG022 + , EDG023 - , EDG024 - .
/END_LOOPS
/EDGES
  EDG009 ; VTX005 , VTX006 ; CRV009 + .
  EDG010 ; VTX006 , VTX007 ; CRV010 + .
  EDG011 ; VTX008 , VTX007 ; CRV011 + .
  EDG012 ; VTX005 , VTX008 ; CRV012 + .
  EDG021 ; VTX009 , VTX010 ; CRV021 + .
  EDG022 ; VTX010 , VTX011 ; CRV022 + .
  EDG023 ; VTX012 , VTX011 ; CRV023 + .
  EDG024 ; VTX009 , VTX012 ; CRV024 + .
/END_EDGES
/VERTICES
  VTX005 ; PT005 .
  VTX006 ; PT006 .
  VTX007 ; PT007 .
  VTX008 ; PT008 .
  VTX009 ; PT009 .
  VTX010 ; PT010 .
  VTX011 ; PT011 .
  VTX012 ; PT012 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR006 ; PLANE ; VEC003 ; 2.00000 .
/END_SURFACES
/CURVES
```

```
  CRV009 ; LINE ; PT005, VEC001 .
  CRV010 ; LINE ; PT006, VEC002 .
  CRV011 ; LINE ; PT007, VEC001 .
  CRV012 ; LINE ; PT008, VEC002 .
  CRV021 ; LINE ; PT009, VEC001 .
  CRV022 ; LINE ; PT010, VEC002 .
  CRV023 ; LINE ; PT011, VEC001 .
  CRV024 ; LINE ; PT012, VEC002 .
/END_CURVES
/POINTS
  PNT005 ; 0.00000, 0.00000, 2.00000 .
  PNT006 ; 4.00000, 0.00000, 2.00000 .
  PNT007 ; 4.00000, 4.00000, 2.00000 .
  PNT008 ; 0.00000, 4.00000, 2.00000 .
  PNT009 ; 1.00000, 1.00000, 2.00000 .
  PNT010 ; 3.00000, 1.00000, 2.00000 .
  PNT011 ; 3.00000, 3.00000, 2.00000 .
  PNT012 ; 1.00000, 3.00000, 2.00000 .
/END_POINTS
/UNIT_VECTORS
  VEC001 ; 1.00000, 0.00000, 0.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
  VEC003 ; 0.00000, 0.00000, 1.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_CHECK_PLANE
/END_FEATURE_TOP_FACE
/FEATURE_POCKET
/SIDE_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC007 ; LOP008; SUR007 + .
/END_FACES
/LOOPS
  LOP008 ; EDG017 + , EDG021 + , EDG018 - , EDG013 - .
/END_LOOPS
/EDGES
  EDG017 ; VTX013 , VTX009 ; CRV017 + .
  EDG021 ; VTX009 , VTX010 ; CRV021 + .
  EDG018 ; VTX014 , VTX010 ; CRV018 + .
  EDG013 ; VTX013 , VTX014 ; CRV013 + .
/END_EDGES
/VERTICES
```

```
  VTX013 ; PT013 .
  VTX009 ; PT009 .
  VTX010 ; PT010 .
  VTX014 ; PT014 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR007 ; PLANE ; VEC002 ; 1.00000 .
/END_SURFACES
/CURVES
  CRV017 ; LINE ; PT013, VEC003 .
  CRV021 ; LINE ; PT009, VEC001 .
  CRV018 ; LINE ; PT014, VEC003 .
  CRV013 ; LINE ; PT013, VEC001 .
/END_CURVES
/POINTS
  PNT013 ; 1.00000, 1.00000, 1.75000 .
  PNT009 ; 1.00000, 1.00000, 2.00000 .
  PNT010 ; 3.00000, 1.00000, 2.00000 .
  PNT014 ; 3.00000, 1.00000, 1.75000 .
/END_POINTS
/UNIT_VECTORS
  VEC003 ; 0.00000, 0.00000, 1.00000 .
  VEC001 ; 1.00000, 0.00000, 0.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_SIDE_PLANE
/SIDE_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC008 ; LOP009; SUR008 - .
/END_FACES
/LOOPS
  LOP009 ; EDG018 + , EDG022 + , EDG019 - , EDG014 - .
/END_LOOPS
/EDGES
  EDG018 ; VTX014 , VTX010 ; CRV018 + .
  EDG022 ; VTX010 , VTX011 ; CRV022 + .
  EDG019 ; VTX015 , VTX011 ; CRV019 + .
  EDG014 ; VTX014 , VTX015 ; CRV014 + .
/END_EDGES
/VERTICES
```

```
  VTX014 ; PT014 .
  VTX010 ; PT010 .
  VTX011 ; PT011 .
  VTX015 ; PT015 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR008 ; PLANE ; VEC001 ; 3.00000 .
/END_SURFACES
/CURVES
  CRV018 ; LINE ; PT014, VEC003 .
  CRV022 ; LINE ; PT010, VEC002 .
  CRV019 ; LINE ; PT015, VEC003 .
  CRV014 ; LINE ; PT014, VEC002 .
/END_CURVES
/POINTS
  PNT014 ; 3.00000, 1.00000, 1.75000 .
  PNT010 ; 3.00000, 1.00000, 2.00000 .
  PNT011 ; 3.00000, 3.00000, 2.00000 .
  PNT015 ; 3.00000, 3.00000, 1.75000 .
/END_POINTS
/UNIT_VECTORS
  VEC003 ; 0.00000, 0.00000, 1.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_SIDE_PLANE
/SIDE_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC009 ; LOP010; SUR009 - .
/END_FACES
/LOOPS
  LOP010 ; EDG020 + , EDG023 + , EDG019 - , EDG015 - .
/END_LOOPS
/EDGES
  EDG020 ; VTX016 , VTX012 ; CRV020 + .
  EDG023 ; VTX012 , VTX011 ; CRV023 + .
  EDG019 ; VTX015 , VTX011 ; CRV019 + .
  EDG015 ; VTX016 , VTX015 ; CRV015 + .
/END_EDGES
/VERTICES
```

```
  VTX016 ; PT016 .
  VTX012 ; PT012 .
  VTX011 ; PT011 .
  VTX015 ; PT015 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR009 ; PLANE ; VEC002 ; 3.00000 .
/END_SURFACES
/CURVES
  CRV020 ; LINE ; PT016, VEC003 .
  CRV023 ; LINE ; PT011, VEC001 .
  CRV019 ; LINE ; PT015, VEC003 .
  CRV015 ; LINE ; PT015, VEC001 .
/END_CURVES
/POINTS
  PNT016 ; 1.00000, 3.00000, 1.75000 .
  PNT012 ; 1.00000, 3.00000, 2.00000 .
  PNT011 ; 3.00000, 3.00000, 2.00000 .
  PNT015 ; 3.00000, 3.00000, 1.75000 .
/END_POINTS
/UNIT_VECTORS
  VEC003 ; 0.00000, 0.00000, 1.00000 .
  VEC001 ; 1.00000, 0.00000, 0.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_SIDE_PLANE
/SIDE_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC010 ; LOP011; SUR010 + .
/END_FACES
/LOOPS
  LOP011 ; EDG017 + , EDG024 + , EDG020 - , EDG016 - .
/END_LOOPS
/EDGES
  EDG017 ; VTX013 , VTX009 ; CRV017 + .
  EDG024 ; VTX009 , VTX012 ; CRV024 + .
  EDG020 ; VTX016 , VTX012 ; CRV020 + .
  EDG016 ; VTX013 , VTX016 ; CRV016 + .
/END_EDGES
/VERTICES
```

```
  VTX013 ; PT013 .
  VTX009 ; PT009 .
  VTX012 ; PT012 .
  VTX016 ; PT016 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR010 ; PLANE ; VEC001 ; 1.00000 .
/END_SURFACES
/CURVES
  CRV017 ; LINE ; PT013, VEC003 .
  CRV024 ; LINE ; PT012, VEC002 .
  CRV020 ; LINE ; PT016, VEC003 .
  CRV016 ; LINE ; PT016, VEC002 .
/END_CURVES
/POINTS
  PNT013 ; 1.00000, 1.00000, 1.75000 .
  PNT009 ; 1.00000, 1.00000, 2.00000 .
  PNT012 ; 1.00000, 3.00000, 2.00000 .
  PNT016 ; 1.00000, 3.00000, 1.75000 .
/END_POINTS
/UNIT_VECTORS
  VEC003 ; 0.00000, 0.00000, 1.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_SIDE_PLANE
/BOTTOM_PLANE
/TOPOLOGY
/SHELLS
/END_SHELLS
/FACES
  FAC011 ; LOP012; SUR011 + .
/END_FACES
/LOOPS
  LOP012 ; EDG013 + , EDG014 + , EDG015 - , EDG016 - .
/END_LOOPS
/EDGES
  EDG013 ; VTX013 , VTX014 ; CRV013 + .
  EDG014 ; VTX014 , VTX015 ; CRV014 + .
  EDG015 ; VTX016 , VTX015 ; CRV015 + .
  EDG016 ; VTX013 , VTX016 ; CRV016 + .
/END_EDGES
/VERTICES
```

```
  VTX013 ; PT013 .
  VTX014 ; PT014 .
  VTX015 ; PT015 .
  VTX016 ; PT016 .
/END_VERTICES
/END_TOPOLOGY
/GEOMETERY
/SURFACES
  SUR011 ; PLANE ; VEC003 ; 1.75000 .
/END_SURFACES
/CURVES
  CRV013 ; LINE ; PT013, VEC001 .
  CRV014 ; LINE ; PT014, VEC002 .
  CRV015 ; LINE ; PT015, VEC001 .
  CRV016 ; LINE ; PT016, VEC002 .
/END_CURVES
/POINTS
  PNT013 ; 1.00000, 1.00000, 1.75000 .
  PNT014 ; 3.00000, 1.00000, 1.75000 .
  PNT015 ; 3.00000, 3.00000, 1.75000 .
  PNT016 ; 1.00000, 3.00000, 1.75000 .
/END_POINTS
/UNIT_VECTORS
  VEC001 ; 1.00000, 0.00000, 0.00000 .
  VEC002 ; 0.00000, 1.00000, 0.00000 .
/END_UNIT_VECTORS
/END_GEOMETRY
/END_BOTTOM_PLANE
/END_FEATURE_POCKET
/END_FEATURE_REMOVAL_MODEL
```

Appendix D (FEATURE_ID Program Code Main Program)

```
PROGRAM FEATURE_ID;

{ Writtem by: Jeff Silkman
  Date:       Fall 1988
  Language:   Pascal - Turbo Pascal 4.0
  Purpose:    Thesis Project - Feature Removal Analysis}

{Program Feature_id is the main controlling program of the
FEATURE_ID system.  This routine generates a main menu on
the CRT from which the user calls to other program modules
while performing a feature removal analysis.

Parent:   None

Chidren:  STORE, SELECT, SET_VIEW, MODEL_IT, RUN, D_FEAT,
          PRINT, CLRSCR, TEXTMODE, TEXTCOLOR

Units:    COMMON, PROG, CRT}

{ ---------------------------------------------------------- }

USES  {units}

  CRT,
  COMMON, PROG;

{ ---------------------------------------------------------- }

BEGIN

  {initialize system}

  COMMAND := 'XXXXX';
  PART_ID := 'UNINITIALIZED';
  V_COM   := 'NOT SET';

  {build screen menu, wait for command}

  WHILE COMMAND <> 'EXIT' DO
  BEGIN

    CLRSCR;
    TEXTMODE(C80 + FONT8X8);

    {some lines are truncated for margin requirements}
```

```
TEXTCOLOR(7);
WRITELN('********************************************');
WRITELN('*                                          *');
WRITELN('*                FEATURES PROGRAM COMMANDS *');
WRITELN('*                                          *');
WRITELN('********************************************');
TEXTCOLOR(2);
WRITELN(' PART = ',PART_ID);
WRITELN;
WRITELN(' VIEW SELECT- ',V_COM,':: HOME X:',ZERO_X,
        'HOME Z:',ZERO_Z,' SCALE:',SCALE_FACTOR);

TEXTCOLOR(6);
WRITELN('****************************************');
WRITELN('*                                      *');
WRITELN('* SOURCE FILE OPERATIONS -             *');
WRITELN('*                                      *');
WRITELN('* STORE  : STORE PART DEFINITION       *');
WRITELN('*                                      *');
WRITELN('* SELECTION FOR DISPLAY -              *');
WRITELN('*                                      *');
WRITELN('* SELECT : SELECT PART                 *');
WRITELN('* VIEW   : SPECIFY DISPLAY VIEW        *');
WRITELN('*                                      *');
WRITELN('* DISPLAY::                            *');
WRITELN('*                                      *');
WRITELN('* MODEL  : WIRE FRAME MODEL            *');
WRITELN('*                                      *');
WRITELN('* ANALYSIS -                           *');
WRITELN('*                                      *');
WRITELN('* RUN    : RUN FEATURE REMOVAL ANALYSIS*');
WRITELN('* D_FEAT : DISPLAY FEATURES            *');
WRITELN('* PRINT  : PRINT RESULTS OF ANALYSIS   *');
WRITELN('*                                      *');
WRITELN('*                                      *');
WRITELN('* PROGRAM -                            *');
WRITELN('*                                      *');
WRITELN('* EXIT   : EXIT PROGRAM                *');
WRITELN('*                                      *');
WRITELN('****************************************');


TEXTCOLOR(2);
```

Appendix D (FEATURE_ID Program Code Main Program)

```
    WRITELN('ENTER PROGRAM COMMAND :');
    READLN(COMMAND);

    IF COMMAND = 'STORE' THEN
    BEGIN
      STORE;
    END;

    IF COMMAND = 'SELECT' THEN
    BEGIN
       SELECT;
    END;
    IF COMMAND = 'VIEW' THEN
    BEGIN
       SET_VIEW;
    END;

    IF COMMAND = 'MODEL' THEN
    BEGIN
       CLRSCR;
       MODEL_IT;
    END;

    IF COMMAND = 'RUN' THEN
    BEGIN
      RUN;
    END;

    IF COMMAND = 'D_FEAT' THEN
    BEGIN
      D_FEAT;
    END;

    IF COMMAND = 'PRINT' THEN
    BEGIN
       PRINT;
    END;

  END; {while}

END. {program FEATURE_ID}
```

Appendix E (FEATURE_ID Program Code Unit Prog)

```
UNIT PROG;

{ Unit PROG contains the procedures called to by the main
program via a menu selection by the user.  Each routine
contained in PROG controls one program module of the
FEATURE_ID system.}

{ ******************************************************* }

INTERFACE

USES {units}

  CRT, PRINTER, GRAPH,
  COMMON, SORT, PLOT, MODEL, READ, TOP_FACE, POCKET;

{subroutines with global access}

PROCEDURE STORE;
PROCEDURE SELECT;
PROCEDURE SET_VIEW;
PROCEDURE MODEL_IT;
PROCEDURE RUN;
PROCEDURE D_FEAT;
PROCEDURE PRINT;

{ ******************************************************* }

IMPLEMENTATION

{local variables to this unit}

TYPE

 VW = STRING[1];                                 {view type}
 FILE_TP = STRING[40];                           {file type}
 LINE_TP = STRING[100];                          {line type}
 FEATURE_FILES = ARRAY[1..10] OF FILE_TP;        {files}

VAR

 VAL_SELECT  : BOOLEAN;                 {valid select flag}
 VAL_VIEW    : BOOLEAN;                 {valid view flag}
 VAL_MOD     : BOOLEAN;                 {valid model flag}
 CONTINUE    : BOOLEAN;
```

```
VAL_D_FEAT   : BOOLEAN;                {valid display features}
PRINT_FILE : STRING[30];               {print file}
PRINT_FILE_ID : STRING[40];            {print file id}
PRINT_LINE : STRING[100];              {print line}

VIEW_TYP : VW;
RESULT_FILE : FILE_TP;
RESULT_PATH : FILE_TP;
LINE        : LINE_TP;
LINES       : LINE_TP;
FILE_TYPE   : FILE_TP;
TYPE_FILE   : FILE_TP;

RESULT   : TEXT;
RES_FILE : TEXT;
IN_FILE  : TEXT;
RES_OUT  : TEXT;
SORT_IN  : TEXT;
INFILE   : TEXT;

NUM_TOP_FACES : INTEGER;
NUM_POCKETS : INTEGER;

TOP_FACES : FEATURE_FILES;
POCKETS   : FEATURE_FILES;

I : INTEGER;
P : POINTER;
SIZE : WORD;


{===========================================================}

PROCEDURE STORE;

{ This routine is used to control the part model
initialization module of the FEATYRE_ID system.  A part
model name is entered by the user in order to initialize a
new input database for use by the rest of the program.

Parent: FEATURE_ID

Children: TEXTCOLOR, CLRSCR, SORT_PART }
```

```
{ ------------------------------------------------------- }

  BEGIN
    TEXTCOLOR(5);
    CLRSCR;
    WRITELN('ENTER PART NAME :');
    READLN(PART_NAME);
    WRITELN;
    CLRSCR;
    SORT_PART;
  END; {store}

{==========================================================}

PROCEDURE SELECT;

{ The SELECT routine is used to select a part model for
analysis.

Parent: FEATURE_ID

Children: CLRSCR, TEXTCOLOR

  BEGIN

    CLRSCR;
    TEXTCOLOR(6);

    WRITELN;
    WRITELN('*****************************');
    WRITELN('*                           *');
    WRITELN('* DISPLAY SELECTION CONTROL *');
    WRITELN('*                           *');
    WRITELN('*****************************');
    WRITELN;

    WRITELN('ENTER PART MAME:');
    READLN(PART_NAME);
    WRITELN;
    VAL_SELECT := TRUE;
    PART_ID := PART_NAME;
    CLRSCR;
  END; {select}
```

Appendix E (FEATURE_ID Program Code Unit Prog)

```
{=============================================================}

PROCEDURE SET_VIEW;

{ The SET_VIEW procedure is used to select viewing
parameters when graphically modeling a part model or the
results of an analysis

Parent: FEATURE_ID

Children: CLRSCR, TEXTCOLOR }

{ -----------------------------------------------------------}

  BEGIN

    CLRSCR;

    WRITELN('****************');
    WRITELN('*              *');
    WRITELN('* VIEW CONTROL *');
    WRITELN('*              *');
    WRITELN('****************');
    WRITELN;

    TEXTCOLOR(10);

    WRITELN('STANDARD VIEWS:');
    WRITELN;
    WRITELN('F - FRONT');
    WRITELN('L - LEFT');
    WRITELN('O - ORTHOG');
    WRITELN('U - USER SPECIFIED');
    WRITELN;
    WRITELN('ENTER VIEW REQUEST LETTER:');
    READLN(VIEW_TYP);
    WRITELN;

    IF VIEW_TYP = 'F' THEN
    BEGIN
      ALPHA := 0;
      BETA := 0;
      GAMMA := 0;
    END;
```

```
IF VIEW_TYP = 'L' THEN
BEGIN
  ALPHA := 90;
  BETA := 0;
  GAMMA := 0;
END;

IF VIEW_TYP = 'O' THEN
BEGIN
  ALPHA := 30;
  BETA := 0;
  GAMMA := 30;
END;

IF VIEW_TYP = 'U' THEN
BEGIN
   WRITELN('ENTER Z ROTATION ANGLE:');
   READLN(ALPHA);
   WRITELN;
   WRITELN('ENTER X ROTATION ANGLE:');
   READLN(BETA);
   WRITELN;
   WRITELN('ENTER Y ROTATION ANGLE:');
   READLN(GAMMA);
   WRITELN;
END;

COORD := FALSE;
WRITELN('INCLUDE COORDINATES IN MODEL : (Y/N)');
READLN(CHOICE);
IF CHOICE = 'Y' THEN
BEGIN
  COORD := TRUE;
END;
IF CHOICE = 'N' THEN
BEGIN
  COORD := FALSE;
END;


WRITELN;
WRITELN(' ORIENT DISPLAY ON SCREEN BY SPECIFYING HOME
         COORDINATES');
WRITELN(' IN RELATION TO SCREEN ORIGIN:');
```

```
    TEXTCOLOR(9);
    WRITELN;
    WRITELN(' SCREEN (0,0,0)  ----------------> +X [640]');
    WRITELN('                 |                         ');
    WRITELN('                 |                         ');
    WRITELN('                 |                         ');
    WRITELN('                 |                         ');
    WRITELN('                 |                         ');
    WRITELN('                 V                         ');
    WRITELN('                                           ');
    WRITELN('               + Z [320]                   ');
    WRITELN;
    WRITELN;

    TEXTCOLOR(10);
    WRITELN('ENTER HOME COORDINATES: ');
    WRITELN;
    WRITELN('ENTER PART ORIGIN LOCATION IN SCREEN X AXIS:');
    READLN(ZERO_X);
    WRITELN;
    WRITELN('ENTER PART ORIGIN LOCATION IN SCREEN Z AXIS:');
    READLN(ZERO_Z);
    WRITELN;
    WRITELN('ENTER SCALE FACTOR :');
    READLN(SCALE_FACTOR);
    WRITELN;

    V_COM := VIEW_TYP;

    VAL_VIEW := TRUE;


    WRITELN('SET VIEW COMPLETE - HIT RETURN TO CONTINUE');

    READLN;

    CLRSCR;

  END; {set_view}

{==========================================================}

PROCEDURE MODEL_IT;

{ The MODEL_IT routine is used to provide a graphic display
```

```
of a selected part model.

Parent: FEATURE_ID

Children: READ_DATA, CONVERT_TO_ORTH_INT,
          CONVERT_TO_ORTH_REAL, DRAW_FRAME, MOD_PLOT_POINTS
          MOD_DRAW_LINES, INITGRAPH, CLOSEGRAPH }

{ ---------------------------------------------------------}

  BEGIN

    VAL_MOD := TRUE;

    IF NOT VAL_SELECT THEN
    BEGIN
      WRITELN;
      WRITELN('YOU MUST RUN SELECT ROUTINE FIRST');
      WRITELN;
      VAL_MOD := FALSE;
    END;

    IF NOT VAL_VIEW THEN
    BEGIN
      WRITELN;
      WRITELN('YOU MUST RUN VIEW ROUTINE FIRST');
      WRITELN;
      VAL_MOD := FALSE;
    END;

    IF VAL_MOD THEN
    BEGIN

      READ_DATA;
      CONVERT_TO_ORTH_REAL;
      CONVERT_TO_ORTH_INT;

      GRAPHDRIVER := 0;
      INITGRAPH(GRAPHDRIVER,GRAPHMODE,'C:/TP');

      DRAW_FRAME;
      COLOR_ID := GREEN;
      MOD_PLOT_POINTS;
      MOD_DRAW_LINES;
```

```
      READLN;

      CLOSEGRAPH;

    END;

  END; {MODEL_IT}

{===========================================================}

PROCEDURE RUN;

{ The RUN procedure is used to control the execution of a
part feature removal analysis.

Parent: FEATURE_ID

Children: FIND_TOP_FACES, FIND_POCKETS }

{-----------------------------------------------------------}

BEGIN


  CLRSCR;
  TEXTCOLOR(14);

  WRITELN('ENTER PART NAME FOR ANALYSIS :');
  READLN(PART_ANAL_ID);
  WRITELN;
  WRITELN('ENTER BLANK NAME FOR ANALYSIS :');
  READLN(BLANK_ANAL_ID);
  WRITELN;
  WRITELN('ENTER ANALYSIS STORAGE NAME :');
  READLN(ANAL_ID);
  CLRSCR;

  WRITELN('********************************************');
  WRITELN('     PART = ',PART_ANAL_ID,'                 ');
  WRITELN('     BLANK = ',BLANK_ANAL_ID,'               ');
  WRITELN('     ANALYSIS_ID = ',ANAL_ID,'               ');
  WRITELN('********************************************');
  WRITELN;
  WRITELN('CHECKING FOR FEATURE "TOP_FACE"');
  WRITELN;
```

```
  FIND_TOP_FACES;
  WRITELN;
  WRITELN('TOP_FACE ANALYSIS COMPLETE - HIT RETURN TO
          CONTINUE');
READLN;

  CLRSCR;
  TEXTCOLOR(14);
  WRITELN('*******************************************');
  WRITELN('     PART = ',PART_ANAL_ID,'                ');
  WRITELN('     BLANK = ',BLANK_ANAL_ID,'              ');
  WRITELN('     ANALYSIS_ID = ',ANAL_ID,'              ');
  WRITELN('*******************************************');
  WRITELN;
  WRITELN('CHECKING FOR FEATURE "POCKET"');
  FIND_POCKETS;
  WRITELN;
  WRITELN('POCKET ANALYSIS COMPLETE - HIT RETURN TO
          CONTINUE');
  READLN;

  CLRSCR;

  {COMBINE REMOVAL FEATURE BASED PART MODEL}

  TEXTCOLOR(14);
  WRITELN('*******************************************');
  WRITELN('     PART = ',PART_ANAL_ID,'                ');
  WRITELN('     BLANK = ',BLANK_ANAL_ID,'              ');
  WRITELN('     ANALYSIS_ID = ',ANAL_ID,'              ');
  WRITELN('*******************************************');
  WRITELN;
  TEXTCOLOR(9);
  WRITELN('CREATING FEATURE REMOVAL FILE ',ANAL_ID);
  WRITELN;

  WRITELN('STORING ',ANAL_ID,' :: TOP_FACE');

  FILE_TYPE := CONCAT(ANAL_ID,'.FBM');
  TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
  ASSIGN(RES_OUT,TYPE_FILE);
  REWRITE(RES_OUT);

  FILE_TYPE := CONCAT(BLANK_ANAL_ID,'.TFC');
  TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
```

```
ASSIGN(SORT_IN,TYPE_FILE);
RESET(SORT_IN);

WRITELN(RES_OUT,'/FEATURE_REMOVAL_MODEL');
WRITELN(RES_OUT,'/HEADER');
WRITELN(RES_OUT,'  PART NAME = ',PART_ANAL_ID);
WRITELN(RES_OUT,'  BLANK NAME = ',BLANK_ANAL_ID);
WRITELN(RES_OUT,'  ANALYSIS NAME = ',ANAL_ID);
WRITELN(RES_OUT,'/END_HEADER');

{STORE TOP_FACE ANALYSIS}

WRITELN(RES_OUT,'/FEATURE_TOP_FACE');

WHILE NOT EOF(SORT_IN) DO
BEGIN
  READLN(SORT_IN,LINE);
  WRITELN(RES_OUT,LINE);
END;

CLOSE(SORT_IN);

FILE_TYPE := CONCAT(PART_ANAL_ID,'.TFC');
TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
ASSIGN(SORT_IN,TYPE_FILE);

RESET(SORT_IN);

WHILE NOT EOF(SORT_IN) DO
BEGIN
  READLN(SORT_IN,LINE);
  WRITELN(RES_OUT,LINE);
END;
CLOSE(SORT_IN);

WRITELN(RES_OUT,'/END_FEATURE_TOP_FACE');


{STORE POCKET ANALYSIS}

WRITELN;
WRITELN('STORING ',ANAL_ID,' :: POCKETS');
WRITELN;
```

```
TYPE_FILE := CONCAT('C:\TP\RESULT\',ANAL_ID);
FILE_TYPE := CONCAT(TYPE_FILE,'.FID');

ASSIGN(SORT_IN,FILE_TYPE);

RESET(SORT_IN);

WHILE NOT EOF(SORT_IN) DO
BEGIN
  READLN(SORT_IN,LINE);
  IF (LINE = '/POCKET') THEN
  BEGIN
    WRITELN(RES_OUT,'/FEATURE_POCKET');

    WHILE (LINE <> '/END_POCKET') DO
    BEGIN
      READLN(SORT_IN,LINE);
      IF (LINE <> '/END_POCKET' ) THEN
      BEGIN
        ASSIGN(INFILE,LINE);
        RESET(INFILE);
        WHILE NOT EOF(INFILE) DO
        BEGIN
          READLN(INFILE,LINES);
          WRITELN(RES_OUT,LINES);
        END;
        CLOSE(INFILE);
      END;
    END;
    WRITELN(RES_OUT,'/END_FEATURE_POCKET');
  END;
END;

CLOSE(SORT_IN);

WRITELN(RES_OUT,'/END_FEATURE_REMOVAL_MODEL');

CLOSE(RES_OUT);

TEXTCOLOR(14);

WRITELN;

WRITELN('ANALYSIS ',ANAL_ID,' COMPLETE - HIT RETURN TO
```

```
          CONTINUE');
  WRITELN;
  READLN;

END; {run}


{==========================================================}

PROCEDURE D_FEAT;

{ The D_FEAT procedure is used to provide graphic display of
the results of a feature removal analysis.

Parent: FEAATURE_ID

Children: READ_DATA, SORT_PART_2, CONVERT_TO_ORTH_INT
          CONVERT_TO_ORTHH_REAL, MOD_PLOT_POINTS,
          DRAW_FEATURE_TRACK, DRAW_FEATURE_FRAME,
          MOD_DRAW_LINES, INITGRAPH, CLOSEGRAPH, CLRSCR,
          TEXTCOLOR          }


{local variable}

VAR

 S : STRING[2];

BEGIN

  CLRSCR;
  TEXTCOLOR(6);
  VAL_D_FEAT := TRUE;

  WRITELN('*****************************');
  WRITELN('*                           *');
  WRITELN('*  FEATURE DISPLAY ROUTINE  *');
  WRITELN('*                           *');
  WRITELN('*****************************');
  WRITELN;
  WRITELN('ENTER ANALYSIS RUN ID NAME :');
  READLN(RUN_ANAL_ID);

  IF NOT VAL_VIEW THEN
```

```
BEGIN
  WRITELN;
  WRITELN('YOU MUST RUN VIEW ROUTINE FIRST');
  WRITELN;
  VAL_D_FEAT := FALSE;
  READLN;
END;

IF VAL_D_FEAT THEN
BEGIN

{GET TOP FACE FILE ID'S}


  RESULT_FILE := CONCAT(RUN_ANAL_ID,'.FID');
  RESULT_PATH := CONCAT('C:\TP\RESULT\',RESULT_FILE);

  ASSIGN(RESULT,RESULT_PATH);

  RESET(RESULT);

  NUM_TOP_FACES := 0;
  I := 1;
  CONTINUE := TRUE;

  WHILE NOT EOF(RESULT) DO
  BEGIN

    READLN(RESULT,LINE);

    IF (LINE = '/TOP_FACE') THEN
    BEGIN

       WHILE (CONTINUE) DO
       BEGIN
          READLN(RESULT,LINE);

          IF (LINE <> '/END_TOP_FACE') THEN
          BEGIN
            TOP_FACES[I] := LINE;
            I := I + 1;
            NUM_TOP_FACES := NUM_TOP_FACES + 1;
          END;
          IF (LINE = '/END_TOP_FACE') THEN
          BEGIN
```

```
              CONTINUE := FALSE
            END;
         END;
      END;

    END;

    CLOSE(RESULT);


  {GET POCKET ID'S}


    RESULT_FILE := CONCAT(RUN_ANAL_ID,'.FID');
    RESULT_PATH := CONCAT('C:\TP\RESULT\',RESULT_FILE);

    ASSIGN(RESULT,RESULT_PATH);

    RESET(RESULT);

    NUM_POCKETS := 0;
    I := 1;
    CONTINUE := TRUE;

    WHILE NOT EOF(RESULT) DO
    BEGIN

      READLN(RESULT,LINE);

      IF (LINE = '/POCKET') THEN
      BEGIN

         WHILE (CONTINUE) DO
         BEGIN
            READLN(RESULT,LINE);

            IF (LINE <> '/END_POCKET') THEN
            BEGIN
              POCKETS[I] := LINE;
              I := I + 1;
              NUM_POCKETS := NUM_POCKETS + 1;
            END;
            IF (LINE = '/END_POCKET') THEN
            BEGIN
```

```
          CONTINUE := FALSE
        END;
      END;
    END;

  END;

  CLOSE(RESULT);


  WRITELN('NUM TOP_FACES = ',NUM_TOP_FACES);
  FOR I := 1 TO NUM_TOP_FACES DO
  BEGIN
    WRITELN(TOP_FACES[I]);
  END;


  { COPY FILE TO \TP\SOURCE\D_FEAT.BRP}
  { SORT, READ, PLOT}

  GRAPHDRIVER := 0;
  INITGRAPH(GRAPHDRIVER,GRAPHMODE,'C:/TP');

  DRAW_FEATURE_FRAME;

  COUNT_DISP := 25;


  FOR I := 1 TO NUM_TOP_FACES DO
  BEGIN

    ASSIGN(RES_FILE,TOP_FACES[I]);
    ASSIGN(IN_FILE,'C:\TP\SOURCE\D_FEAT.BRP');

    RESET(RES_FILE);
    REWRITE(IN_FILE);

    WHILE NOT EOF(RES_FILE) DO
    BEGIN
      READLN(RES_FILE,LINE);
      WRITELN(IN_FILE,LINE);
    END;

    CLOSE(RES_FILE);
    CLOSE(IN_FILE);
```

```
    PART_NAME := 'D_FEAT';

    SORT_PART_2;

    READ_DATA;

    CONVERT_TO_ORTH_REAL;
    CONVERT_TO_ORTH_INT;

    COLOR_ID := GREEN;

    STR(I,S);

    DRAW_FEATURE := CONCAT('TOP FACE: PL',S);
    DRAW_FEATURE_TRACK;

    MOD_PLOT_POINTS;
    MOD_DRAW_LINES;
    READLN;

  END;

  FOR I := 1 TO NUM_POCKETS DO
  BEGIN

    ASSIGN(RES_FILE,POCKETS[I]);
    ASSIGN(IN_FILE,'C:\TP\SOURCE\D_FEAT.BRP');

    RESET(RES_FILE);
    REWRITE(IN_FILE);

    WHILE NOT EOF(RES_FILE) DO
    BEGIN
      READLN(RES_FILE,LINE);
      WRITELN(IN_FILE,LINE);
    END;

    CLOSE(RES_FILE);
    CLOSE(IN_FILE);


    PART_NAME := 'D_FEAT';
```

```
      SORT_PART_2;

      READ_DATA;

      CONVERT_TO_ORTH_REAL;
      CONVERT_TO_ORTH_INT;

      COLOR_ID := CYAN;

      STR(I,S);

      DRAW_FEATURE := CONCAT('POCKET: PL',S);
      DRAW_FEATURE_TRACK;

      MOD_PLOT_POINTS;
      MOD_DRAW_LINES;
      READLN;

    END;

    CLOSEGRAPH;

  END;

END;  {d_feat}

{===========================================================}

PROCEDURE PRINT;

{ The print procedure is used to provide hard copies of
input databases as well as output feature removal model
databases.}

Parent : FEATURE_ID

Children : CLRSCR, TEXTCOLOR

{local program variable}

VAR

  RESPONSE : STRING[1];
  PRINT_PART_ID : STRING[8];
```

Appendix E (FEATURE_ID Program Code Unit Prog)

```
  INPUT : TEXT;

{ --------------------------------------------------------}

BEGIN

  CLRSCR;
  TEXTCOLOR(7);
  WRITELN('**********************');
  WRITELN('*                    *');
  WRITELN('*  PRINT ROUTINE     *');
  WRITELN('*                    *');
  WRITELN('**********************');

  WRITELN('TURN ON THE PRINTER !!!!!!!');


  WRITELN;
  WRITELN;

  WRITELN(' CHOOSE OPTION:');
  WRITELN;
  WRITELN('A: PRINT PART/BLANK SOURCE FILE');
  WRITELN('B: PRINT FEATURE ANALYSIS RESULT SUMMARY');
  WRITELN('C: ABORT ... ESPECIALLY IF YOU HAVE NO
           PRINTER !');
  WRITELN;
  READLN(RESPONSE);
  WRITELN;

  IF (RESPONSE = 'A') THEN
  BEGIN
    WRITELN('ENTER PART OR BLANK NAME: ');
    WRITELN;
    READLN(PRINT_PART_ID);

    PRINT_FILE := CONCAT('C:\TP\SOURCE\',PRINT_PART_ID);
    PRINT_FILE_ID := CONCAT(PRINT_FILE,'.BRP');

    ASSIGN(INPUT,PRINT_FILE_ID);

    RESET(INPUT);

    WRITELN('PRINTING ',PRINT_FILE_ID);
```

```
    WRITELN;

    WRITELN(LST);
    WRITELN(LST);

    WRITELN(LST,'BOUNDRY REPRESENTAION DATA FILE FOR');
    WRITELN(LST,PRINT_FILE_ID);
    WRITELN(LST);
    WRITELN(LST,'*************************************');

    WHILE NOT EOF(INPUT) DO
    BEGIN
       READLN(INPUT,PRINT_LINE);
       WRITELN(LST,PRINT_LINE);
    END;

    WRITELN;
    WRITELN('DONE PRINTING - HIT RETURN TO CONTINUE');
    READLN;

  END;


  IF (RESPONSE = 'B') THEN
  BEGIN
    WRITELN('ENTER ANALYSIS ID NAME: ');
    WRITELN;
    READLN(PRINT_PART_ID);

    PRINT_FILE := CONCAT('C:\TP\RESULT\',PRINT_PART_ID);
    PRINT_FILE_ID := CONCAT(PRINT_FILE,'.FID');

    ASSIGN(INPUT,PRINT_FILE_ID);

    RESET(INPUT);

    WRITELN('PRINTING ',PRINT_FILE_ID);
    WRITELN;

    WRITELN(LST);
    WRITELN(LST);


    WRITELN(LST,'FEATURE STORAGE FILES FOR');
```

```
WRITELN(LST,PRINT_FILE_ID);
WRITELN(LST);
WRITELN(LST,'*************************************');

WHILE NOT EOF(INPUT) DO
BEGIN
   READLN(INPUT,PRINT_LINE);
   WRITELN(LST,PRINT_LINE);
END;

WRITELN;


PRINT_FILE := CONCAT('C:\TP\RESULT\',PRINT_PART_ID);
PRINT_FILE_ID := CONCAT(PRINT_FILE,'.FBM');

ASSIGN(INPUT,PRINT_FILE_ID);

RESET(INPUT);

WRITELN('PRINTING ',PRINT_FILE_ID);
WRITELN;

FOR I := 1 TO 7 DO
BEGIN

  WRITELN(LST);
END;

WRITELN(LST,'FEATURE BASED DATA FILE FOR');
WRITELN(LST,PRINT_FILE_ID);
WRITELN(LST);
WRITELN(LST,'*************************************');

WHILE NOT EOF(INPUT) DO
BEGIN
   READLN(INPUT,PRINT_LINE);
   WRITELN(LST,PRINT_LINE);
END;

WRITELN;
WRITELN('DONE PRINTING - HIT RETURN TO CONTINUE');
READLN;
```

```
  END;
END; {print}
END. {unit PROG}
```

Appendix F (FEATURE_ID Program Code Unit Common)

```
UNIT COMMON;

{  The  COMMON unit is a global common  variable  definition
area.   By  using  Turbo Unit capability,  program  variable
definitions  could  be made within this  unit  and  accessed
throughout the program. }

{ ********************************************************}

INTERFACE

{-------------------------------------------------------}

{ GLOBAL COMMON VARIABLES }

{*** FOR UNIT PROG ***}

TYPE

  COM_STR = STRING[8];           {user command type}
  ID_STR  = STRING[30];          {part model display type}
  VIEW_STR = STRING[60];         {system setup display type}

VAR

  COMMAND  : COM_STR;            {command from user}
  PART_ID  : ID_STR;             {part model selected}
  BLANK_ID : ID_STR;             {blank model selected}

  PART_ANAL_ID   :  ID_STR;      {part name for analysis}
  BLANK_ANAL_ID : ID_STR;        {blank name for analysis}
  RUN_ANAL_ID   : ID_STR;        {analysis save name}
  ANAL_ID       : ID_STR;        {analysis result name}

  CHOICE        : COM_STR;       {command chosen}

  GRAPHDRIVER : INTEGER;         {spec for unit graph}
  GRAPHMODE   : INTEGER;         {spec for unit graph}
  V_COM       : VIEW_STR;        {view option chosen}

  DRAW_FEATURE : STRING[15];     {feature type drawn}
  COUNT_DISP   : INTEGER;        {number of features}
  COLOR_ID     : WORD;           {color code}

  COORD  :   BOOLEAN;            {include coords. in model}
```

Appendix F (FEATURE_ID Program Code Unit Common)

```
  AXIS   : BOOLEAN;               {include axis in model}

{*** FOR UNIT SORT ***}

TYPE

  STRG_TYPE = STRING[50];         {string type}

VAR

  PART_NAME : STRG_TYPE;          {part name}
  PART_DIR  : STRG_TYPE;          {part directory}


{*** FOR UNIT ARRAYS ***}

TYPE

  VECTOR = RECORD                 {record for vectors}
            UX1 : REAL;           {unit vectors}
            UY1 : REAL;
            UZ1 : REAL;
           END;

  POINT  = RECORD                 {record for points}
            X1 : REAL;            {point coords.}
            Y1 : REAL;
            Z1 : REAL;
           END;

  ORTHOG = RECORD                 {record for projection}
            X2 : REAL;            {projection coords.}
            Y2 : REAL;
            Z2 : REAL;
           END;


  ORTHOG_GR = RECORD              {record for pixel proj}
               X3 : INTEGER;      {pixel proj coords.}
               Y3 : INTEGER;
               Z3 : INTEGER;
              END;


  CURVE = RECORD                  {record for curves}
```

```
            CRV_TYPE    : STRING[4]; {curve type}
            CRV_PNT_NUM : INTEGER;   {curve point ref.}
            CRV_VEC_NUM : INTEGER;   {curve vector ref.}
           END;


  SURFACE = RECORD                  {surface record}
              SURF_TYPE    : STRING[5]; {surface type}
              SURF_VEC_NUM : INTEGER;   {surface vector ref.}
              SURF_DELTA   : REAL;      {surface distance}
             END;

  VERTEX = RECORD                   {record for vertices}
             VERT_PNT_NUM : INTEGER;   {point ref.}
            END;


  EDGE = RECORD                     {record for edges}
           EDG_VERT_NUM_1 : INTEGER;  {point 1 ref.}
           EDG_VERT_NUM_2 : INTEGER;  {point 2 ref.}
           EDG_CRV_NUM    : INTEGER;  {curve ref.}
           EDG_SENSE      : INTEGER;  {edge sense}
          END;


  LOOP = RECORD                     {record for loop}
           LOOP_EDG_NUMS  : ARRAY[1..25] OF INTEGER;
                            {loop edge number refs.}
           LOOP_SENSE     : ARRAY[1..25] OF INTEGER;
                            {loop sense}
          END;

  FACE = RECORD                     {record for face}
           FAC_LOOP_NUMS  : ARRAY[1..25] OF INTEGER;
                            {face loop number refs.}
           FAC_SURF_NUM   : INTEGER; {face surface ref.}
           FAC_SENSE      : INTEGER; {face sense}
          END;


  SHELL = RECORD                    {record for shell}
            SHL_FAC_NUM   : ARRAY[1..50] OF INTEGER;
                            {shell face number refs.}
           END;
```

Appendix F (FEATURE_ID Program Code Unit Common)

```
  VECTOR_ARRAY =    ARRAY[1..50] OF VECTOR;   {vector array}
  POINT_ARRAY =     ARRAY[1..50] OF POINT;    {point array}
  ORTHOG_ARRAY =    ARRAY[1..50] OF ORTHOG;   {proj array}
  ORTHOG_GR_ARRAY = ARRAY[1..50] OF ORTHOG_GR; {pixel array}
  CURVE_ARRAY =     ARRAY[1..50] OF CURVE;    {curve array}
  SURFACE_ARRAY =   ARRAY[1..50] OF SURFACE;  {surface array}
  VERTEX_ARRAY =    ARRAY[1..50] OF VERTEX;   {vertex array}
  EDGE_ARRAY =      ARRAY[1..50] OF EDGE;     {edge array}
  LOOP_ARRAY =      ARRAY[1..50] OF LOOP;     {loop array}
  FACE_ARRAY =      ARRAY[1..50] OF FACE;     {face array}
  SHELL_ARRAY =     ARRAY[1..50] OF SHELL;    {shell array}

  POINTLNS =        ARRAY[1..50] OF INTEGER;  {point line array}

VAR

  VECTORS     : VECTOR_ARRAY;     {vectors}
  POINTS      : POINT_ARRAY;      {points}
  ORTHOGS     : ORTHOG_ARRAY;     {projections}
  ORTHOG_GRS  : ORTHOG_GR_ARRAY;  {pixel projections}
  CURVES      : CURVE_ARRAY;      {curves}
  SURFACES    : SURFACE_ARRAY;    {surfaces}
  VERTICEES   : VERTEX_ARRAY;     {vertices}
  EDGES       : EDGE_ARRAY;       {edges}
  LOOPS       : LOOP_ARRAY;       {loops}
  FACES       : FACE_ARRAY;       {faces}
  SHELLS      : SHELL_ARRAY;      {shells}

  POINT_LN    : POINTLNS;         {line number of point}

  ARRAY_LOC : INTEGER;            {array counter}
  NUMPOINTS : INTEGER;            {number of points}
  NUMVECS   : INTEGER;            {number of vectors}
  NUMCURVS  : INTEGER;            {number of curves}
  NUMSURFS  : INTEGER;            {number of surfaces}
  NUMVERTS  : INTEGER;            {number of vertices}
  NUMEDGES  : INTEGER;            {number of edges}
  NUMLOOPS  : INTEGER;            {number of loops}
  NUMFACES  : INTEGER;            {number of faces}
  NUMSHELLS : INTEGER;            {number of shells}

  POINT_LN_NUM : INTEGER;         {number of lines for points}

  POINT_ID : INTEGER;             {point identifier}
```

Appendix F (FEATURE_ID Program Code Unit Common)

```
{*** FOR UNIT PLOT ***}

VAR

  VIEW_X : REAL;                  {view temp}
  VIEW_Y : REAL;
  VIEW_Z : REAL;

  ZERO_X : INTEGER;               {home pixel}
  ZERO_Y : INTEGER;
  ZERO_Z : INTEGER;

  ALPHA        : REAL;            {rotation angles}
  BETA         : REAL;
  GAMMA        : REAL;
  SCALE_FACTOR : REAL;            {scale factor}

    {----------------------------------------------------}

PROCEDURE DUMMY;

{this procedure does nothing. It completes syntax
requirements.

{----------------------------------------------------------}

IMPLEMENTATION

PROCEDURE DUMMY;

BEGIN

END; {dummy}

END. {unit common}
```

Appendix G (FEATURE_ID Program Code Unit SORT)

```
UNIT SORT;

{Unit SORT contains the routines used to initialize a part
or part blank model.  These routines break up larger part
model database files into smaller files used by FEATURE_ID.}

{*********************************************************}

INTERFACE

USES  {units}

  CRT,
  COMMON;

{subroutines with global access}

PROCEDURE SORT_PART;
PROCEDURE SORT_PART_2;

{---------------------------------------------------------}

IMPLEMENTATION

{local variables to this unit}

TYPE

  LINE   = STRING[100];         {line type}
  FILE_NAME = STRING[50];       {file name type}

VAR

  UNIT_FILE   : STRG_TYPE;      {input file}
  ENTITY      : STRG_TYPE;      {dictionary type}
  OUT_TYPE    : STRG_TYPE;      {dictionary file type}
  START_IDENT: STRG_TYPE;       {start dictionary id}
  END_IDENT   : STRG_TYPE;      {end of dictionary id}
  BAD_CALL    : BOOLEAN;        {bad call flag}

{=========================================================}

PROCEDURE SPLIT;
```

Appendix G (FEATURE_ID Program Code Unit SORT)

```
{This routine is used to build a small file containing data
between the starting and ending identifier of a dictionary
type within a part model.

Parent: SORT_PART

Children: TEXTMODE, CLRSCR}

{local procedure variables}

TYPE

  LINE      = STRING[100];       {line type declaration}
  FILE_NAME = STRING[50];        {file name type declaration}

VAR

  TYPE_OUT    : TEXT;             {file type output}
  BREP_IN     : TEXT;             {BREP model input}
  TYPE_FILE   : FILE_NAME;        {file type}
  TYPE_LINE   : LINE;             {line type}
  I           : INTEGER;          {counter}

{ -------------------------------------------------------}
BEGIN

  BAD_CALL := FALSE;
  TYPE_FILE := CONCAT(PART_DIR,OUT_TYPE);

  ASSIGN(TYPE_OUT,TYPE_FILE);
  ASSIGN(BREP_IN,UNIT_FILE);

  {$I-}
  RESET(BREP_IN);     {report file open error}
  {$I+}

  IF IORESULT <> 0 THEN
  BEGIN
    WRITELN('NO FILE OF PART ON DISK');
    BAD_CALL := TRUE;
    WRITELN;
  END;

  IF NOT BAD_CALL THEN
  BEGIN
```

```
      REWRITE(TYPE_OUT);
      WHILE NOT EOF(BREP_IN) DO
      BEGIN
         READLN(BREP_IN,TYPE_LINE);
         IF (TYPE_LINE = START_IDENT) THEN
         BEGIN
           READLN(BREP_IN,TYPE_LINE);
           I := 0;
           TEXTCOLOR(12);
           WHILE TYPE_LINE <> END_IDENT DO
           BEGIN
              WRITELN(TYPE_LINE);
              WRITELN(TYPE_OUT,TYPE_LINE);
              IF TYPE_LINE[LENGTH(TYPE_LINE)] = '.' THEN
              BEGIN
                 I := I + 1;
              END;
              READLN(BREP_IN,TYPE_LINE);
           END;
         WRITELN;
         WRITELN('TOTAL OF ',I,' ',ENTITY,' FOR ',PART_NAME);
         WRITELN;
         TEXTCOLOR(3);
         WRITELN('HIT RETURN TO CONTINUE');
         READLN;
         CLRSCR;
         TEXTCOLOR(2);
       END;
     END;
  CLOSE(TYPE_OUT);
  CLOSE(BREP_IN);
  END;
END; {split}

{==========================================================}

PROCEDURE SORT_PART;

{ This routine is used to set identifiers for each
dictionary type to be stored in smaller files.  Also, calls
to the SPLIT routine create the files.

Parent: STORE

Children: TEXTCOLOR, CLRSCR, SPLIT }
```

Appendix G (FEATURE_ID Program Code Unit SORT)

```
{ -----------------------------------------------------------}

BEGIN

   TEXTMODE(C80 + FONT8x8);
   CLRSCR;

   PART_DIR := CONCAT('C:\TP\SOURCE\',PART_NAME);
   UNIT_FILE := CONCAT(PART_DIR,'.BRP');

   {initialize vector file}
   ENTITY := 'VECTORS';
   OUT_TYPE := '.VEC';
   START_IDENT := '/UNIT_VECTORS';
   END_IDENT := '/END_UNIT_VECTORS';
   SPLIT;

   {initialize point file}
   ENTITY := 'POINTS';
   OUT_TYPE := '.PNT';
   START_IDENT := '/POINTS';
   END_IDENT := '/END_POINTS';
   SPLIT;

   {initialize curve file}
   ENTITY := 'CURVES';
   OUT_TYPE := '.CRV';
   START_IDENT := '/CURVES';
   END_IDENT := '/END_CURVES';
   SPLIT;

   {initialize surface file}
   ENTITY := 'SURFACES';
   OUT_TYPE := '.SUR';
   START_IDENT := '/SURFACES';
   END_IDENT := '/END_SURFACES';
   SPLIT;

   {initialize vertices file}
   ENTITY := 'VERTICES';
   OUT_TYPE := '.VTX';
   START_IDENT := '/VERTICES';
   END_IDENT := '/END_VERTICES';
   SPLIT;
```

```
    {initialize edge file}
    ENTITY := 'EDGES';
    OUT_TYPE := '.EDG';
    START_IDENT := '/EDGES';
    END_IDENT := '/END_EDGES';
    SPLIT;

    {initialize loop file}
    ENTITY := 'LOOPS';
    OUT_TYPE := '.LOP';
    START_IDENT := '/LOOPS';
    END_IDENT := '/END_LOOPS';
    SPLIT;

    {initialize faces file}
    ENTITY := 'FACES';
    OUT_TYPE := '.FAC';
    START_IDENT := '/FACES';
    END_IDENT := '/END_FACES';
    SPLIT;

    {initialize shell file}
    ENTITY := 'SHELLS';
    OUT_TYPE := '.SHL';
    START_IDENT := '/SHELLS';
    END_IDENT := '/END_SHELLS';
    SPLIT;

    WRITELN;
    WRITELN('FINISHED INITIALIZING ',UNIT_FILE);
    WRITELN;
    WRITELN('READY');
    WRITELN;

    NORMVIDEO;

END; {sort_part}

{==========================================================}

PROCEDURE SPLIT_2;

{This routine is identical in function to SPLIT, except it
is tailored to initialize output feature removal models for
use by the D_FEAT routine
```

Appendix G (FEATURE_ID Program Code Unit SORT)

Parent: SORT_PART_2

Children: TEXTMODE, CLRSCR, SPLIT_2

```
{local procedure variables}

TYPE

  LINE   = STRING[100];
  FILE_NAME = STRING[50];

VAR

  TYPE_OUT    : TEXT;
  BREP_IN     : TEXT;
  TYPE_FILE   : FILE_NAME;
  TYPE_LINE   : LINE;
  I           : INTEGER;

{ -----------------------------------------------------------}

BEGIN

  BAD_CALL := FALSE;
  TYPE_FILE := CONCAT(PART_DIR,OUT_TYPE);
  ASSIGN(TYPE_OUT,TYPE_FILE);
  ASSIGN(BREP_IN,UNIT_FILE);

  {$I-}
  RESET(BREP_IN);
  {$I+}

  IF IORESULT <> 0 THEN
  BEGIN
    WRITELN('NO FILE OF PART ON DISK');
    BAD_CALL := TRUE;
    WRITELN;
  END;

  IF NOT BAD_CALL THEN
  BEGIN
    REWRITE(TYPE_OUT);
    WHILE NOT EOF(BREP_IN) DO
    BEGIN
       READLN(BREP_IN,TYPE_LINE);
```

```
       IF (TYPE_LINE = START_IDENT) THEN
       BEGIN
         READLN(BREP_IN,TYPE_LINE);
         I := 0;
         WHILE TYPE_LINE <> END_IDENT DO
         BEGIN
            WRITELN(TYPE_OUT,TYPE_LINE);
            IF TYPE_LINE[LENGTH(TYPE_LINE)] = '.' THEN
            BEGIN
               I := I + 1;
            END;
            READLN(BREP_IN,TYPE_LINE)
         END;
       END;
    END;
    CLOSE(TYPE_OUT);
    CLOSE(BREP_IN);
  END;
END; {split_2}

{===========================================================}

PROCEDURE SORT_PART_2;

{This routine is identical in function to the SORT_PART
routine. It is tailored to initialize the feature based
output database for use by the D_FEAT routine.

Parent: D_FEAT

Children: TEXTMODE, CLRSCR, SPLIT_2  }

{ ---------------------------------------------------------}

BEGIN

   PART_DIR := CONCAT('C:\TP\SOURCE\',PART_NAME);
   UNIT_FILE := CONCAT(PART_DIR,'.BRP');

   ENTITY := 'VECTORS';
   OUT_TYPE := '.VEC';
   START_IDENT := '/UNIT_VECTORS';
   END_IDENT := '/END_UNIT_VECTORS';
   SPLIT_2;
```

Appendix G (FEATURE_ID Program Code Unit SORT)

```
ENTITY := 'POINTS';
OUT_TYPE := '.PNT';
START_IDENT := '/POINTS';
END_IDENT := '/END_POINTS';
SPLIT_2;

ENTITY := 'CURVES';
OUT_TYPE := '.CRV';
START_IDENT := '/CURVES';
END_IDENT := '/END_CURVES';
SPLIT_2;

ENTITY := 'SURFACES';
OUT_TYPE := '.SUR';
START_IDENT := '/SURFACES';
END_IDENT := '/END_SURFACES';
SPLIT_2;

ENTITY := 'VERTICES';
OUT_TYPE := '.VTX';
START_IDENT := '/VERTICES';
END_IDENT := '/END_VERTICES';
SPLIT_2;

ENTITY := 'EDGES';
OUT_TYPE := '.EDG';
START_IDENT := '/EDGES';
END_IDENT := '/END_EDGES';
SPLIT_2;

ENTITY := 'LOOPS';
OUT_TYPE := '.LOP';
START_IDENT := '/LOOPS';
END_IDENT := '/END_LOOPS';
SPLIT_2;

ENTITY := 'FACES';
OUT_TYPE := '.FAC';
START_IDENT := '/FACES';
END_IDENT := '/END_FACES';
SPLIT_2;

ENTITY := 'SHELLS';
OUT_TYPE := '.SHL';
START_IDENT := '/SHELLS';
```

```
   END_IDENT := '/END_SHELLS';
   SPLIT_2;

END; {split_2}


END. {unit sort}
```

Appendix H (FEATURE_ID Code Unit PLOT)

```
UNIT PLOT;

{ Unit PLOT is used to compute the projection of the actual
three dimensional part model coordinates to the two
dimensional screen display coordinates.  Proc.
CONVERT_TO_ORTH_REAL converts to real number values, Proc.
CONVERT_TO_ORTH_INT converts real values into integer values
which correlate to pixel locations.                          }

{ ********************************************************}

INTERFACE

USES {units}

  COMMON;

{subroutines with global access}

PROCEDURE CONVERT_TO_ORTH_REAL;
PROCEDURE CONVERT_TO_ORTH_INT;

{ --------------------------------------------------------}

IMPLEMENTATION

(local varibles to this unit}

VAR
   I   : INTEGER;        { counter }
   J   : INTEGER;        { counter }
   K   : INTEGER;        { counter }

   ALPHA_RAD : REAL;     { alpha angle in radians}
   BETA_RAD  : REAL;     { beta angle in readians}
   GAMMA_RAD : REAL;     { gamma angle in radians}

{ ========================================================}

PROCEDURE CONVERT_TO_ORTH_REAL;

{This routine is used to convert the part model cartesian
coordinates into two dimensional projection coordinates for
use in modeling the part graphically.
```

Appendix H (FEATURE_ID Code Unit PLOT)

Parents: MODEL_IT, D_FEAT

Children: none }

```
{ ---------------------------------------------------------}

BEGIN

  ALPHA_RAD := ALPHA*PI/180.0;
  BETA_RAD := BETA*PI/180.0;
  GAMMA_RAD := GAMMA*PI/180.0;

  FOR I := 1 TO NUMPOINTS DO
  BEGIN

  {Compute orthogonal projection as specified by the user}

  {HEADING: Z-AXIS ROTATION --> ANGLE ALPHA}

    ORTHOGS[I].X2 := POINTS[I].Y1*SIN(ALPHA_RAD)
                     + POINTS[I].X1*COS(ALPHA_RAD);

    ORTHOGS[I].Y2 := POINTS[I].Y1*COS(ALPHA_RAD)
                     - POINTS[I].X1*SIN(ALPHA_RAD);

    ORTHOGS[I].Z2 := POINTS[I].Z1;

  {PITCH: X AXIS ROTATION --> ANGLE BETA}

    ORTHOGS[I].X2 := ORTHOGS[I].X2*COS(BETA_RAD)
                     - ORTHOGS[I].Z2*SIN(BETA_RAD);

    ORTHOGS[I].Y2 := ORTHOGS[I].Y2;

    ORTHOGS[I].Z2 := ORTHOGS[I].Z2*COS(BETA_RAD)
                     + ORTHOGS[I].X2*SIN(BETA_RAD);

  {BANK: Y AXIS ROTATION --> ANGLE GAMMA}

    ORTHOGS[I].X2 := ORTHOGS[I].X2;

    ORTHOGS[I].Y2 := ORTHOGS[I].Y2*COS(GAMMA_RAD)
```

```
                     + ORTHOGS[I].Z2*SIN(GAMMA_RAD);

    ORTHOGS[I].Z2 := ORTHOGS[I].Z2*COS(GAMMA_RAD)
                     - ORTHOGS[I].Y2*SIN(GAMMA_RAD);

  END; {FOR}
END; {convert_to_orth_real}

{ ==========================================================}

PROCEDURE CONVERT_TO_ORTH_INT;

{ For each converted point, use the user supplied scale
factor and calculate the integer (pixel) coordinates for the
model display.

Parents: MODEL_IT, D_FEAT

Children: none

{ ----------------------------------------------------------}
BEGIN

  FOR I := 1 TO NUMPOINTS DO
  BEGIN

    ORTHOG_GRS[I].X3 := ZERO_X +
               ROUND(ORTHOGS[I].X2*100.0/SCALE_FACTOR);

    ORTHOG_GRS[I].Y3 :=
                ROUND(ORTHOGS[I].Y2*100.0/SCALE_FACTOR);


    ORTHOG_GRS[I].Z3 := ZERO_Z -
                ROUND(ORTHOGS[I].Z2*100.0/SCALE_FACTOR);

  END; {for}

END; {convert_to_orth_int}

END. {unit plot}
```

Appendix I (FEATURE_ID Program Code Unit MODEL)

```
UNIT MODEL;

{Unit PLOT contains graphics routines used in plotting and
drawing graphic representations of part, part blank, and
feature of removal databases.  )

{**********************************************************}

INTERFACE

USES {units}

  CRT, GRAPH,
  COMMON;

{subroutines with global access}

PROCEDURE MOD_PLOT_POINTS;
PROCEDURE MOD_DRAW_LINES;
PROCEDURE DRAW_FRAME;
PROCEDURE DRAW_FEATURE_FRAME;
PROCEDURE DRAW_FEATURE_TRACK;

{----------------------------------------------------------}

IMPLEMENTATION

{local variables to this unit}

VAR
   I : INTEGER;  {counters}
   J : INTEGER;
   K : INTEGER;

{**********************************************************}

PROCEDURE MOD_PLOT_POINTS;

{This procedure is used to plot the points of a model.

 Parents: MODEL_IT, D_FEAT

 Children: PUTPIXEL, SETCOLOR, SETTEXTSTYLE, STR, OUTTEXT }

{----------------------------------------------------------}
```

```
VAR
 COORD_STR : STRING[20];
 CX : STRING[5];
 CY : STRING[5];
 CZ : STRING[5];

BEGIN
  IF NOT COORD THEN   {just plot points}
  BEGIN
   FOR I := 1 TO NUMPOINTS DO
   BEGIN
     PUTPIXEL(ORTHOG_GRS[I].X3,ORTHOG_GRS[I].Z3,5);
   END;
  END;

  IF COORD THEN         {include coordinates while plotting}
  BEGIN
    FOR I := 1 TO NUMPOINTS DO
    BEGIN
      PUTPIXEL(ORTHOG_GRS[I].X3,ORTHOG_GRS[I].Z3,WHITE);

      SETCOLOR(WHITE);
      SETTEXTSTYLE(SMALLFONT, HORIZDIR, 3);

      STR(POINTS[I].X1,CX);
      STR(POINTS[I].Y1,CY);
      STR(POINTS[I].Z1,CZ);

      CX := CONCAT(CX,',');
      CX := CONCAT(CY,',');

      COORD_STR := CONCAT(CX,CY);
      COORD_STR := CONCAT(COORD_STR,CZ);

      OUTTEXTXY(ORTHOG_GRS[I].X3,ORTHOG_GRS[I].Z3,
                COORD_STR);
    END;
  END;

END; {mod_plot_points}

{===========================================================}

PROCEDURE MOD_DRAW_LINES;
```

Appendix I (FEATURE_ID Program Code Unit MODEL)

```
{ This routine is used to draw lines to create the part
image of the selected model.

Parents: MODEL_IT, D_FEAT

Children: SETCOLOR, LINE  }

{ ---------------------------------------------------------}

BEGIN

   FOR  I := 1 TO NUMEDGES DO
   BEGIN

     SETCOLOR(COLOR_ID);

     LINE(ORTHOG_GRS[POINT_LN[EDGES[I].EDG_VERT_NUM_1]].X3,
          ORTHOG_GRS[POINT_LN[EDGES[I].EDG_VERT_NUM_1]].Z3,
          ORTHOG_GRS[POINT_LN[EDGES[I].EDG_VERT_NUM_2]].X3,
          ORTHOG_GRS[POINT_LN[EDGES[I].EDG_VERT_NUM_2]].Z3);
   END;

END; {mod_draw_lines}

{ =========================================================}

PROCEDURE DRAW_FRAME;

{This routine draws a reference frame to the screen before
plotting begins.

Parent: MODEL_IT

Children: SETCOLOR, SETLINESTYLE, LINE, SETTEXTSTYLE,
          SETTEXTJUSTIFY, OUTTEXTXY  }

{ -------------------------------------------------------}

BEGIN;

    SETCOLOR(4);
    SETLINESTYLE(SOLIDLN, 0, THICKWIDTH);

    LINE(1,1,639,1);
    LINE(639,1,639,319);
```

```
    LINE(639,319,1,319);
    LINE(1,319,1,1);
    LINE(550,1,550,319);

    SETLINSTYLE(SOLIDLINE, 0, NORMWIDTH);

    LINE(1,15,550,15);

    SETCOLOR(WHITE);

    SETTEXTSTYLE(SMALLFONT, HORIZDIR, 4);
    OUTTEXTXY(ZERO_X, ZERO_Z, '(0,0,0)');
    PUTPIXEL(ZERO_X, ZERO_Z, WHITE);
    OUTTEXTXY(15,310,'HIT RETURN AFTER EACH PLANE IS
              PLOTTED');

    SETTEXTSTYLE(SANSSERIFFONT, HORIZDIR, 1);
    SETTEXTJUSTIFY(LEFTTEXT, BOTTOMTEXT);
    OUTTEXTXY(5, 14, PART_NAME);

END; {draw_frame}

{============================================================}

PROCEDURE DRAW_FEATURE_FRAME;

{ This procedure is used to draw a reference frame to the
screen while plotting with the D_FEAT command.

Parent: D_FEAT

Children: SETCOLOR, SETLINESTYLE, LINE, PUTPIXEL,
          GETTEXTSTYLE, SETTEXTJUSTIFY, OUTTEXTXY }

{ ----------------------------------------------------------}

BEGIN;

    SETCOLOR(4);

    SETLINESTYLE(SOLIDLN, 0, THICKWIDTH);

    LINE(1,1,639,1);
    LINE(639,1,639,319);
    LINE(639,319,1,319);
```

```
     LINE(1,319,1,1);
     LINE(550,1,550,319);

     SETLINESTYLE(SOLIDLN, 0, NORMWIDTH);

     LINE(1,15,550,15);

     SETCOLOR(WHITE);

     SETTEXTSTYLE(SMALLFONT, HORIZDIR, 4);
     OUTTEXTXY(ZERO_X, ZERO_Z, '(0,0,0)');
     PUTPIXEL(ZERO_X, ZERO_Z, WHITE);
     OUTTEXTXY(15,310,'HIT RETURN AFTER EACH PLANE IS
           PLOTTED');
     OUTTEXTXY(551, 10, 'FEATURE BY');
     OUTTEXTXY(560, 16, 'COLOR');

     SETTEXTSTYLE(SANSSERIFFONT, HORIZDIR, 1);
     SETTEXTJUSTIFY(LEFTTEXT, BOTTOMTEXT);
     OUTTEXTXY(5, 14, RUN_ANAL_ID);

END; {draw_feature_frame}

{=========================================================}

PROCEDURE DRAW_FEATURE_TRACK;

{This routine is used to provide a color coded
identification message to the user as each feature component
is drawn to the screen while in D_FEAT.

Parent: D_FEAT

Children: SETCOLOR, SETTEXTJUSTIFY, OUTTEXTXY, SETTEXTSTYLE}

{ -------------------------------------------------------}
BEGIN
    COUNT_DISP := COUNT_DISP + 10;
    SETCOLOR(COLOR_ID);
    SETTEXTSTYLE(SMALLFONT, HORIZDIR, 4);
    SETTEXTJUSTIFY(LEFTTEXT, BOTTOMTEXT);
    OUTTEXTXY(552, COUNT_DISP, DRAW_FEATURE);
END; {draw_feature_track}

END. {unit model}
```

Appendix J (FEATURE_ID Program COde Unit READ)

```
UNIT READ;

{Unit READ is used to read data from the disk files
associated with a particular part model into the program.}

{*********************************************************}

INTERFACE

USES {units}

  COMMON, ARRAYS;


{subroutine with global access}

PROCEDURE READ_DATA;

{---------------------------------------------------------}

IMPLEMENTATION

{=========================================================}

PROCEDURE READ_DATA;

{This routine calls to the individual read routines for each
data dictionary type.

Parents: MODEL_IT, RUN, D_FEAT

Children: READ_SHELLS, READ_FACES, READ_LOOPS , READ_EDGES,
          READ_VERTICES, READ_VECTORS, READ_POINTS,
          READ_CURVES, READ_SURFACES    }

{ --------------------------------------------------------

  BEGIN

    PART_DIR := CONCAT('C:\TP\SOURCE\',PART_NAME);

    WRITELN;
    WRITELN('READING DATA');
    WRITELN;
```

```
    READ_VECTORS;
    READ_POINTS;
    READ_CURVES;
    READ_SURFACES;
    READ_VERTICES;
    READ_EDGES;
    READ_LOOPS;
    READ_FACES;
    READ_SHELLS;

  END; {read_data}

END. {unit read}
```

Appendix K (FEATURE_ID Program Code Unit TOP_FACE)

```
UNIT TOP_FACE;

{The TOP_FACE unit contains the routines which perform the
feature removal analysis for the top_face feature.  These
routines are accessed by the RUN program module during a
feature removcal analysis.}

{*********************************************************}

INTERFACE

USES {units}

  CRT,
  COMMON, READ;

{subroutine with global access}

PROCEDURE FIND_TOP_FACES;

{ -------------------------------------------------------}

IMPLEMENTATION

{local variable declarations}

TYPE

  STR1 = STRING[3];              {string types}
  STR2 = STRING[15];
  STR3 = STRING[25];
  STR4 = STRING[100];


  RES_IDS = ARRAY[1..20] OF INTEGER; {analysis result type}

VAR

  ID_NUM_STR : STR1;             {ID number string}
  SORT_FILE  : STR3;             {file to sort}
  TYPE_FILE  : STR3;             {file type}
  FILE_TYPE  : STR2;             {file name}
  LINE       : STR4;             {line string}

  CNT_LOOP   : INTEGER;          {counter}
```

```
  ID_NUM      : INTEGER;            {ID number}
  RES_VEC     : INTEGER;            {resulting vector}
  RES_SURF    : INTEGER;            {resulting surface}
  RES_FACE_NUM : RES_IDS;           {resulting face numbers}
  RES_LOOP_NUM : RES_IDS;           {resulting loop numbers}
  RES_EDGE_NUM : RES_IDS;           {resulting edge numbers}
  RES_VERT_NUM : RES_IDS;           {resulting vertices numbers}
  RES_SURF_NUM : RES_IDS;           {resulting surface numbers}
  RES_CURVE_NUM : RES_IDS;          {resulting curve numbers}
  RES_POINT_NUM : RES_IDS;          {reulting point numbers}
  RES_VECTOR_NUM : RES_IDS;         {resulting vector numbers}

  RES_FACE_CNT : INTEGER;           {face counter}
  RES_LOOP_CNT : INTEGER;           {loop counter}
  RES_EDGE_CNT : INTEGER;           {edge counter}
  RES_VERT_CNT : INTEGER;           {vertices counter}
  RES_SURF_CNT : INTEGER;           {surface counter}
  RES_CURVE_CNT : INTEGER;          {curve counter}
  RES_POINT_CNT : INTEGER;          {point counter}
  RES_VECTOR_CNT : INTEGER;         {vector counter}

  RES_OUT     : TEXT;               {output file}
  SORT_IN     : TEXT;               {input file}

  I : INTEGER;                      {counters}
  J : INTEGER;
  K : INTEGER;
  TEMP : INTEGER;
  A : INTEGER;
  B : INTEGER;
  CODE : INTEGER;

  CONTINUE : BOOLEAN;               {logical for continue}

{=======================================================}

PROCEDURE STORE_TFC_DATA;

{This routine is used to write top face feature data to a
file.  For each analysis, all top face data is stored on
disk as it is collected.

Parent: FIND_TOP_FACES

Children: TEXTCOLOR  }
```

Appendix K (FEATURE_ID Program Code Unit TOP_FACE)

```
{ -------------------------------------------------------}

BEGIN

 { WRITE DATA TO FILE }

 { FACE_DATA : }

     TEXTCOLOR(12);
     WRITELN('SAVING FACE DATA :: TOP_FACE');
     WRITELN(RES_OUT,'/TOPOLOGY');
     WRITELN(RES_OUT,'/SHELLS');
     WRITELN(RES_OUT,'/END_SHELLS');
     WRITELN(RES_OUT,'/FACES');

     SORT_FILE := CONCAT(PART_NAME,'.FAC');
     SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
     ASSIGN(SORT_IN,SORT_FILE);

     FOR I := 1 TO RES_FACE_CNT DO
     BEGIN
       RESET(SORT_IN);
       FOR J := 1 TO NUMFACES DO
       BEGIN
          READLN(SORT_IN,LINE);
          ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
          VAL(ID_NUM_STR,ID_NUM,CODE);
          IF (ID_NUM = RES_FACE_NUM[I]) THEN
          BEGIN
            WRITELN(RES_OUT,LINE);
          END;
       END;
       CLOSE(SORT_IN);
     END;
     WRITELN(RES_OUT,'/END_FACES');

  {LOOP_DATA :}

     WRITELN('SAVING LOOP DATA :: TOP_FACE');
     WRITELN(RES_OUT,'/LOOPS');

     SORT_FILE := CONCAT(PART_NAME,'.LOP');
     SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
     ASSIGN(SORT_IN,SORT_FILE);
```

```
     FOR I := 1 TO RES_LOOP_CNT DO
     BEGIN
       RESET(SORT_IN);
       FOR J := 1 TO NUMLOOPS DO
       BEGIN
          READLN(SORT_IN,LINE);
          ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
          VAL(ID_NUM_STR,ID_NUM,CODE);
          IF (ID_NUM = RES_LOOP_NUM[I]) THEN
          BEGIN
            WRITELN(RES_OUT,LINE);
          END;
       END;
       CLOSE(SORT_IN);
     END;
     WRITELN(RES_OUT,'/END_LOOPS');

  {EDGE_DATA :}

     WRITELN('SAVING EDGE DATA :: TOP_FACE');
     WRITELN(RES_OUT,'/EDGES');

     SORT_FILE := CONCAT(PART_NAME,'.EDG');
     SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
     ASSIGN(SORT_IN,SORT_FILE);

     FOR I := 1 TO RES_EDGE_CNT DO
     BEGIN
       RESET(SORT_IN);
       FOR J := 1 TO NUMEDGES DO
       BEGIN
          READLN(SORT_IN,LINE);
          ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
          VAL(ID_NUM_STR,ID_NUM,CODE);
          IF (ID_NUM = RES_EDGE_NUM[I]) THEN
          BEGIN
            WRITELN(RES_OUT,LINE);
          END;
       END;
       CLOSE(SORT_IN);
     END;
     WRITELN(RES_OUT,'/END_EDGES');

  {VERTICES DATA : }
```

```
    WRITELN('SAVING VERTICES DATA :: TOP_FACE');
    WRITELN(RES_OUT,'/VERTICES');

    SORT_FILE := CONCAT(PART_NAME,'.VTX');
    SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
    ASSIGN(SORT_IN,SORT_FILE);

    FOR I := 1 TO RES_VERT_CNT DO
    BEGIN
      RESET(SORT_IN);
      FOR J := 1 TO NUMVERTS DO
      BEGIN
         READLN(SORT_IN,LINE);
         ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
         VAL(ID_NUM_STR,ID_NUM,CODE);
         IF (ID_NUM = RES_VERT_NUM[I]) THEN
         BEGIN
           WRITELN(RES_OUT,LINE);
         END;
      END;
      CLOSE(SORT_IN);
    END;
    WRITELN(RES_OUT,'/END_VERTICES');

 {SURFACE DATA :}

    WRITELN('SAVING SURFACE DATA :: TOP_FACE');
    WRITELN(RES_OUT,'/END_TOPOLOGY');
    WRITELN(RES_OUT,'/GEOMETERY');
    WRITELN(RES_OUT,'/SURFACES');

    SORT_FILE := CONCAT(PART_NAME,'.SUR');
    SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
    ASSIGN(SORT_IN,SORT_FILE);
    RESET(SORT_IN);

    FOR I := 1 TO NUMSURFS DO
    BEGIN
      READLN(SORT_IN,LINE);
      ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
      VAL(ID_NUM_STR,ID_NUM,CODE);
      IF (ID_NUM = RES_SURF) THEN
      BEGIN
        WRITELN(RES_OUT,LINE);
      END;
```

```
     END;
     WRITELN(RES_OUT,'/END_SURFACES');
     CLOSE(SORT_IN);

  {CURVE_DATA}
     WRITELN('SAVING CURVE DATA :: TOP_FACE');
     WRITELN(RES_OUT,'/CURVES');

     SORT_FILE := CONCAT(PART_NAME,'.CRV');
     SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
     ASSIGN(SORT_IN,SORT_FILE);

     FOR I := 1 TO RES_CURVE_CNT DO
     BEGIN
       RESET(SORT_IN);
       FOR J := 1 TO NUMCURVS DO
       BEGIN
          READLN(SORT_IN,LINE);
          ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
          VAL(ID_NUM_STR,ID_NUM,CODE);
          IF (ID_NUM = RES_CURVE_NUM[I]) THEN
          BEGIN
            WRITELN(RES_OUT,LINE);
          END;
       END;
       CLOSE(SORT_IN);
     END;
     WRITELN(RES_OUT,'/END_CURVES');

  {POINTS DATA}

     WRITELN('SAVING POINT DATA :: TOP_FACE');
     WRITELN(RES_OUT,'/POINTS');

     SORT_FILE := CONCAT(PART_NAME,'.PNT');
     SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
     ASSIGN(SORT_IN,SORT_FILE);

     FOR I := 1 TO RES_POINT_CNT DO
     BEGIN
       RESET(SORT_IN);
       FOR J := 1 TO NUMPOINTS DO
       BEGIN
          READLN(SORT_IN,LINE);
          ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
```

```
          VAL(ID_NUM_STR,ID_NUM,CODE);
          IF (ID_NUM = RES_POINT_NUM[I]) THEN
          BEGIN
            WRITELN(RES_OUT,LINE);
          END;
        END;
        CLOSE(SORT_IN);
      END;
      WRITELN(RES_OUT,'/END_POINTS');

  {VECTOR DATA :}

      WRITELN('SAVING VECTOR DATA :: TOP_FACE');
      WRITELN(RES_OUT,'/UNIT_VECTORS');

      SORT_FILE := CONCAT(PART_NAME,'.VEC');
      SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);
      ASSIGN(SORT_IN,SORT_FILE);
      RESET(SORT_IN);

      FOR I := 1 TO NUMVECS DO
      BEGIN

        READLN(SORT_IN,LINE);
        ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
        VAL(ID_NUM_STR,ID_NUM,CODE);
        IF (ID_NUM = RES_VEC) THEN
        BEGIN
           WRITELN(RES_OUT,LINE);
        END;
        CONTINUE := TRUE;
        J := 1;
        WHILE (J < NUMCURVS) AND (CONTINUE)  DO
        BEGIN
          IF(ID_NUM = CURVES[J].CRV_VEC_NUM) AND
             (ID_NUM <> RES_VEC) THEN
      BEGIN
            CONTINUE := FALSE;
            WRITELN(RES_OUT,LINE);
          END;
          J := J + 1;
        END;
      END;
      WRITELN(RES_OUT,'/END_UNIT_VECTORS');
      WRITELN(RES_OUT,'/END_GEOMETRY');
```

```
      CLOSE(SORT_IN);

      IF (CNT_LOOP = 1) THEN
      BEGIN
         WRITELN(RES_OUT,'/END_ENTRY_PLANE');
         CLOSE(RES_OUT);
      END;

      IF (CNT_LOOP = 2) THEN
      BEGIN
         WRITELN(RES_OUT,'/END_CHECK_PLANE');
         CLOSE(RES_OUT);
      END;

END; {store_tfc_data}

{===========================================================}

PROCEDURE FIND_TOP_FACES;

{This procedure is used to perform the analysis between the
part model and part blank model input databases in order to
collect the data needed to describe the top face feature of
removal.

Parent: RUN

Children: READ_DATA, TEXTCOLOR, STORE_TFC_DATA }

{-----------------------------------------------------------}

BEGIN

   FOR CNT_LOOP := 1 TO 2 DO
   BEGIN
     IF CNT_LOOP = 1 THEN
     BEGIN
       TEXTCOLOR(10);
       WRITELN;
       WRITELN('ANALYSING ',BLANK_ANAL_ID,' :: TOP FACE');
       PART_NAME := BLANK_ANAL_ID;
       READ_DATA;
       FILE_TYPE := CONCAT(BLANK_ANAL_ID,'.TFC');
       TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
```

```
      ASSIGN(RES_OUT,TYPE_FILE);
      REWRITE(RES_OUT);
      WRITELN(RES_OUT,'/ENTRY_PLANE');
    END;

    IF CNT_LOOP = 2 THEN
    BEGIN
      TEXTCOLOR(10);
      WRITELN;
      WRITELN('ANALYSING ',PART_ANAL_ID,' :: TOP FACE');
      PART_NAME := PART_ANAL_ID;
      READ_DATA;
      FILE_TYPE := CONCAT(PART_ANAL_ID,'.TFC');
      TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
      ASSIGN(RES_OUT,TYPE_FILE);
      REWRITE(RES_OUT);
      WRITELN(RES_OUT,'/CHECK_PLANE');
    END;

    RES_FACE_CNT := 0;
    RES_LOOP_CNT := 0;
    RES_EDGE_CNT := 0;
    RES_VERT_CNT := 0;
    RES_SURF_CNT := 0;
    RES_CURVE_CNT := 0;
    RES_POINT_CNT := 0;
    RES_VECTOR_CNT := 0;

{FIND FEATURE ELEMENTS}

    FOR I := 1 TO NUMVECS DO
    BEGIN
      IF (VECTORS[I].UX1 = 0) AND (VECTORS[I].
      UY1 = 0) AND (VECTORS[I].UZ1 = 1) THEN
      BEGIN
         RES_VEC := I;
      END; {IF}
    END; {FOR}

     { GET SURFACE }

    RES_SURF := 0;
    FOR I := 1 TO NUMSURFS DO
    BEGIN
      IF (SURFACES[I].SURF_VEC_NUM = RES_VEC) THEN
```

```
    BEGIN
      IF (SURFACES[I].SURF_DELTA > RES_SURF) THEN
      BEGIN
         RES_SURF := I;
      END; {IF}
    END; {IF}
  END; {FOR}

  { GET DATA }

  A := 1;
  B := 1;

  FOR I := 1 TO NUMFACES DO         {FACES}
  BEGIN
    IF (FACES[I].FAC_SURF_NUM = RES_SURF) THEN
    BEGIN
      RES_FACE_NUM[A] := I;
      RES_FACE_CNT := RES_FACE_CNT + 1;
      A := A + 1;
    END;
  END;

  A := 1;
  FOR I := 1 TO RES_FACE_CNT DO     {LOOPS}
  BEGIN
    FOR J := 1 TO 25 DO
    BEGIN
      IF FACES[RES_FACE_NUM[I]].FAC_LOOP_NUMS[J] > 0 THEN
      BEGIN
        RES_LOOP_NUM[A] :=
          FACES[RES_FACE_NUM[I]].FAC_LOOP_NUMS[J];
        A := A + 1;
        RES_LOOP_CNT := RES_LOOP_CNT + 1;
      END;
    END;
  END;

  A := 1;
  FOR I := 1 TO RES_LOOP_CNT DO     {EDGES}
  BEGIN
    FOR J := 1 TO 25 DO
    BEGIN
      IF LOOPS[RES_LOOP_NUM[I]].LOOP_EDG_NUMS[J] > 0 THEN
      BEGIN
```

```
          RES_EDGE_NUM[A] := LOOPS
            [RES_LOOP_NUM[I]].LOOP_EDG_NUMS[J];
          A := A + 1;
          RES_EDGE_CNT := RES_EDGE_CNT + 1;
        END;
      END;
    END;

    A := 1;
    FOR I := 1 TO RES_EDGE_CNT DO      {VERTICES, CURVES}
    BEGIN
       RES_VERT_NUM[A] :=
           EDGES[RES_EDGE_NUM[I]].EDG_VERT_NUM_1;
       RES_VERT_NUM[A+1] :=
           EDGES[RES_EDGE_NUM[I]].EDG_VERT_NUM_2;
       RES_CURVE_NUM[B] :=
           EDGES[RES_EDGE_NUM[I]].EDG_CRV_NUM;
       B := B + 1;
       A := A + 2;
       RES_VERT_CNT := RES_VERT_CNT + 2;
       RES_CURVE_CNT := RES_CURVE_CNT + 1;
    END;


     {ELIMINATE DOUBLES}

    FOR I := 1 TO RES_VERT_CNT DO
    BEGIN

      TEMP := RES_VERT_NUM[I];
      FOR J := I+1 TO RES_VERT_CNT DO
      BEGIN
        IF RES_VERT_NUM[J] = TEMP THEN
        BEGIN
          RES_VERT_NUM[J] := -111
        END;
      END;
    END;

    { SURFACE = RES_SURF}


    A := 1;
    FOR I := 1 TO RES_VERT_CNT DO     {POINTS}
    BEGIN
```

```
        FOR J := 1 TO 50 DO
        BEGIN
          IF J = RES_VERT_NUM[I] THEN
          BEGIN
            RES_POINT_NUM[A] := VERTICEES[J].VERT_PNT_NUM;
          END;
        END;
        A := A + 1;
        RES_POINT_CNT := RES_POINT_CNT + 1;
      END;

        { VECTOR = RES_VEC}

      STORE_TFC_DATA;

    END;

   {STORE FEATURE IN FEATURE IDENT FILE}

    PART_NAME := ANAL_ID;
    FILE_TYPE := CONCAT(ANAL_ID,'.FID');
    TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
    ASSIGN(RES_OUT,TYPE_FILE);
    REWRITE(RES_OUT);

    WRITELN(RES_OUT,'/TOP_FACE');

    FILE_TYPE := CONCAT(BLANK_ANAL_ID,'.TFC');
    TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
    WRITELN(RES_OUT,TYPE_FILE);

    FILE_TYPE := CONCAT(PART_ANAL_ID,'.TFC');
    TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);
    WRITELN(RES_OUT,TYPE_FILE);

    WRITELN(RES_OUT,'/END_TOP_FACE');

    CLOSE(RES_OUT);

  END; {find_top_faces}

END. {unit top_face}
```

Appendix L (FEATURE_ID Program Code Unit POCKET)

```
UNIT POCKET;

{ The POCKET unit containd the routines used to analyse part
and part blank input databases and determine if any pocket
features of removal are present.  The routines also store
any features found in a result file.}

{ ********************************************************}

INTERFACE

USES {units}

  CRT,
  COMMON, READ;

{subroutines with global access}

PROCEDURE FIND_POCKETS;

{--------------------------------------------------------}

IMPLEMENTATION

{local unit variables}

TYPE

  STR1 = STRING[8];                  {string types}
  STR2 = STRING[30];
  STR3 = STRING[25];
  STR4 = STRING[100];

  RES_IDS = ARRAY[1..20] OF INTEGER;    {resultant ID array}
  TEMP_ARRAY = ARRAY[1..50] OF INTEGER; {temporary array}

VAR

  ID_NUM_STR : STR1;                 {line number string}
  SORT_FILE  : STR3;                 {sort file name}
  TYPE_FILE  : STR3;                 {file type}
  FILE_TYPE  : STR2;                 {file name}
  LINE       : STR4;                 {line def.}
  SUFFIX1    : STR1;                 {file name suffix IDs}
  SUFFIX2    : STR1;                 { "" }
```

Appendix L (FEATURE_ID Program Code Unit POCKET)

```
SUFFIX3     : STR1;              { "" }
FACE_TYPE   : STR1;              {face number ID}
FACE_NUM    : STR1;              {face number}
FEAT_NUM    : STR1;              {feature number}
TRACK_FILE  : STR2;              {file tracking file name}

FEAT_NUMBER  : INTEGER;          {feature number}
FEAT_CNT    : INTEGER;           {feature counter}
CNT_LOOP    : INTEGER;           {loop counter}
ID_NUM      : INTEGER;           {ID number}
RES_VEC     : INTEGER;           {resulting vector}
RES_SURF    : INTEGER;           {resulting surface}
FACE_NUMBER : INTEGER;           {face number}

RES_FACE_NUM : RES_IDS;          {resulting face numbers}
RES_LOOP_NUM : RES_IDS;          {resulting loop numbers}
RES_EDGE_NUM : RES_IDS;          {resulting edge numbers}
RES_VERT_NUM : RES_IDS;          {resulting vertices numbers}
RES_SURF_NUM : RES_IDS;          {resulting surface numbers}
RES_CURVE_NUM : RES_IDS;         {resulting curve numbers}
RES_POINT_NUM : RES_IDS;         {resulting point numbers}
RES_VEC_NUM : RES_IDS;           {resulting vector numbers}

RES_FACE_CNT : INTEGER;          {face counter}
RES_LOOP_CNT : INTEGER;          {loop counter}
RES_EDGE_CNT : INTEGER;          {edge counter}
RES_VERT_CNT : INTEGER;          {vertices counter}
RES_SURF_CNT : INTEGER;          {surface counter}
RES_CURVE_CNT : INTEGER;         {curve counter}
RES_POINT_CNT : INTEGER;         {point counter}
RES_VEC_CNT : INTEGER;           {vector counter}

RES_OUT     : TEXT;              {output file}
SORT_IN     : TEXT;              {input file}
ID_FILE     : TEXT;              {ID file}

I,J,K,x : INTEGER;               {counters}
A,B,C,D,E,F,G,H : INTEGER;
AA,BB,CC,DD,EE,FF,GG,HH,II : INTEGER;

CODE : INTEGER;                  {temp counters}
TEMP : INTEGER;
TEMP_EDGE : RES_IDS;
TEMP_SIDE_LOOP : RES_IDS;        {temp side loops}
SIDE_FACE : RES_IDS;             {side faces}
```

```
   NUM_TEMP_EDGES : INTEGER;      {temp edge counter}
   NUM_TEMP_SIDE_LOOP : INTEGER;{temp side loop counter}
   SIDE_FACE_CNT : INTEGER;       {side face counter}
   BOTTOM_CHECK : TEMP_ARRAY;     {bottom face check}
   NUM_CHECK_EDGES : INTEGER;     {number of edges to check}
   CHECK_COUNT : INTEGER;
   BOTTOM_LOOP : INTEGER;         {resulting bottom loop}
   BOTTOM_FACE : INTEGER;         {resulting bottom face}

   POCKET_FOUND : BOOLEAN;        {logical if pocket found}
   CONTINUE : BOOLEAN;            {logical continue}

{=============================================================}

PROCEDURE CONSOLIDATE;

{ This routine is used to organize resultant data from an
analysis before saving to file.

Parent: FIND_POCKETS

Children: none    }

{ -----------------------------------------------------------}

BEGIN

   RES_FACE_CNT := 0;
   RES_LOOP_CNT := 0;
   RES_EDGE_CNT := 0;
   RES_VERT_CNT := 0;
   RES_SURF_CNT := 0;
   RES_CURVE_CNT := 0;
   RES_POINT_CNT := 0;
   RES_VEC_CNT := 0;


   {FACES}

   A := 1;
   B := 1;

   FOR II := 1 TO NUMFACES DO
   BEGIN
     IF (II = FACE_NUMBER) THEN
```

```
    BEGIN
      RES_FACE_NUM[A] := II;
      RES_FACE_CNT := RES_FACE_CNT + 1;
      RES_SURF := FACES[II].FAC_SURF_NUM;
      A := A + 1;
    END;
  END;

  {LOOPS}

  A := 1;
  FOR II := 1 TO RES_FACE_CNT DO
  BEGIN
    FOR J := 1 TO 25 DO
    BEGIN
      IF FACES[RES_FACE_NUM[II]].FAC_LOOP_NUMS[J] > 0 THEN
      BEGIN
        RES_LOOP_NUM[A] :=
           FACES[RES_FACE_NUM[II]].FAC_LOOP_NUMS[J];
        A := A + 1;
        RES_LOOP_CNT := RES_LOOP_CNT + 1;
      END;
    END;
  END;

  A := 1;
  FOR II := 1 TO RES_LOOP_CNT DO    {EDGES}
  BEGIN
    FOR J := 1 TO 25 DO
    BEGIN
      IF LOOPS[RES_LOOP_NUM[II]].LOOP_EDG_NUMS[J] > 0 THEN
      BEGIN
        RES_EDGE_NUM[A] := LOOPS
          [RES_LOOP_NUM[II]].LOOP_EDG_NUMS[J];
        A := A + 1;
        RES_EDGE_CNT := RES_EDGE_CNT + 1;
      END;
    END;
  END;

  A := 1;
  FOR II := 1 TO RES_EDGE_CNT DO    {VERTICES, CURVES}
  BEGIN
    RES_VERT_NUM[A] :=
       EDGES[RES_EDGE_NUM[II]].EDG_VERT_NUM_1;
```

```
    RES_VERT_NUM[A+1] :=
       EDGES[RES_EDGE_NUM[II]].EDG_VERT_NUM_2;
    RES_CURVE_NUM[B] :=
       EDGES[RES_EDGE_NUM[II]].EDG_CRV_NUM;
    RES_VEC_NUM[B] :=
       CURVES[RES_CURVE_NUM[B]].CRV_VEC_NUM;
   B := B + 1;
    A := A + 2;
    RES_VERT_CNT := RES_VERT_CNT + 2;
    RES_CURVE_CNT := RES_CURVE_CNT + 1;
    RES_VEC_CNT := RES_VEC_CNT + 1;
  END;

  {ELIMINATE DOUBLES IN VERTICES}

  FOR II := 1 TO RES_VERT_CNT DO
  BEGIN

    TEMP := RES_VERT_NUM[II];

    FOR J := II+1 TO RES_VERT_CNT DO
    BEGIN
      IF RES_VERT_NUM[J] = TEMP THEN
      BEGIN
        RES_VERT_NUM[J] := -111
      END;
    END;
  END;

  TEMP := RES_VERT_CNT;
  A := 1;
  RES_VERT_CNT := 0;

  FOR II := 1 TO TEMP DO
  BEGIN
    IF (RES_VERT_NUM[II] > 0) THEN
    BEGIN
      RES_VERT_NUM[A] := RES_VERT_NUM[II];
      A := A + 1;
      RES_VERT_CNT := RES_VERT_CNT + 1;
    END;
  END;

  {ELIMINATE DOUBLES IN VECTORS}
```

```
    FOR II := 1 TO RES_VEC_CNT DO
    BEGIN

      TEMP := RES_VEC_NUM[II];

      FOR J := II+1 TO RES_VEC_CNT DO
      BEGIN

        IF RES_VEC_NUM[J] = TEMP THEN
        BEGIN
          RES_VEC_NUM[J] := -111
        END;
      END;
    END;


    TEMP := RES_VEC_CNT;
    A := 1;
    RES_VEC_CNT := 0;

    FOR II := 1 TO TEMP DO
    BEGIN
      IF (RES_VEC_NUM[II] > 0) THEN
      BEGIN
        RES_VEC_NUM[A] := RES_VEC_NUM[II];
        A := A + 1;
        RES_VEC_CNT := RES_VEC_CNT + 1;
      END;
    END;


    {POINTS}

    A := 1;
    FOR II := 1 TO RES_VERT_CNT DO
    BEGIN
      IF (RES_VERT_NUM[II] > 0) THEN
      BEGIN
        RES_POINT_NUM[A] := RES_VERT_NUM[II];
        A := A + 1;
        RES_POINT_CNT := RES_POINT_CNT + 1;
      END;
    END;

END; {consolodate}
```

Appendix L (FEATURE_ID Program Code Unit POCKET)

```
{--------------------------------------------------------}

PROCEDURE STORE_POC_DATA;

{This routine is used to store the resulting pocket feature
of removal analysis data to output files.

Parent: FIND_POCKETS

Children: TEXTCOLOR

BEGIN

   WRITELN;
   WRITELN('POCKET ',I,' PLANE ',FACE_TYPE,'
           NUMBER ',FEAT_NUMBER);
  WRITELN;

 { WRITE DATA TO FILE }

   SUFFIX1 := CONCAT(FEAT_NUM,'.P');
   SUFFIX2 := CONCAT(SUFFIX1,FACE_TYPE);
   SUFFIX3 := CONCAT(SUFFIX2,FACE_NUM);

   FILE_TYPE := CONCAT(PART_ANAL_ID,SUFFIX3);
   TYPE_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);

   FILE_TYPE := CONCAT(ANAL_ID,'.FID');
   TRACK_FILE := CONCAT('C:\TP\RESULT\',FILE_TYPE);

   ASSIGN(ID_FILE,TRACK_FILE);
   APPEND(ID_FILE);

   IF (FEAT_NUMBER = 1) THEN
   BEGIN
     WRITELN(ID_FILE,'/POCKET');
     WRITELN(ID_FILE,TYPE_FILE);
   END;
   IF (FEAT_NUMBER > 1) AND (FACE_TYPE <> 'B') THEN
   BEGIN
     WRITELN(ID_FILE,TYPE_FILE);
   END;
   IF (FACE_TYPE = 'B') THEN
   BEGIN
```

```
    WRITELN(ID_FILE,TYPE_FILE);
    WRITELN(ID_FILE,'/END_POCKET');
  END;

  CLOSE(ID_FILE);

  ASSIGN(RES_OUT,TYPE_FILE);
  REWRITE(RES_OUT);

  IF (FACE_TYPE = 'S') THEN
  BEGIN
    WRITELN(RES_OUT,'/SIDE_PLANE');
  END;
  IF (FACE_TYPE = 'B' ) THEN
  BEGIN
    WRITELN(RES_OUT,'/BOTTOM_PLANE');
  END;

{ FACE_DATA : }

  TEXTCOLOR(12);
  WRITELN('SAVING FACE DATA :: POCKET ',I,FACE_TYPE);

  WRITELN(RES_OUT,'/TOPOLOGY');
  WRITELN(RES_OUT,'/SHELLS');
  WRITELN(RES_OUT,'/END_SHELLS');
  WRITELN(RES_OUT,'/FACES');

  SORT_FILE := CONCAT(PART_NAME,'.FAC');
  SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

  ASSIGN(SORT_IN,SORT_FILE);

  FOR II := 1 TO RES_FACE_CNT DO
  BEGIN
    RESET(SORT_IN);
    FOR J := 1 TO NUMFACES DO
    BEGIN
      READLN(SORT_IN,LINE);
      ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
      VAL(ID_NUM_STR,ID_NUM,CODE);
      IF (ID_NUM = RES_FACE_NUM[II]) THEN
      BEGIN
        WRITELN(RES_OUT,LINE);
      END;
```

```
      END;
      CLOSE(SORT_IN);
    END;

    WRITELN(RES_OUT,'/END_FACES');

   {LOOP_DATA :}

    WRITELN('SAVING LOOP DATA :: POCKET ',I,FACE_TYPE);
    WRITELN(RES_OUT,'/LOOPS');

    SORT_FILE := CONCAT(PART_NAME,'.LOP');
    SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

    ASSIGN(SORT_IN,SORT_FILE);

    FOR II := 1 TO RES_LOOP_CNT DO
    BEGIN
      RESET(SORT_IN);
      FOR J := 1 TO NUMLOOPS DO
      BEGIN
        READLN(SORT_IN,LINE);
        ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
        VAL(ID_NUM_STR,ID_NUM,CODE);
        IF (ID_NUM = RES_LOOP_NUM[II]) THEN
        BEGIN
          WRITELN(RES_OUT,LINE);
        END;
      END;
      CLOSE(SORT_IN);
    END;

    WRITELN(RES_OUT,'/END_LOOPS');

   {EDGE_DATA :}

    WRITELN('SAVING EDGE DATA :: POCKET ',I,FACE_TYPE);
    WRITELN(RES_OUT,'/EDGES');

    SORT_FILE := CONCAT(PART_NAME,'.EDG');
    SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

    ASSIGN(SORT_IN,SORT_FILE);

    FOR II := 1 TO RES_EDGE_CNT DO
```

```
  BEGIN
    RESET(SORT_IN);
    FOR J := 1 TO NUMEDGES DO
    BEGIN
      READLN(SORT_IN,LINE);
      ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
      VAL(ID_NUM_STR,ID_NUM,CODE);
      IF (ID_NUM = RES_EDGE_NUM[II]) THEN
      BEGIN
        WRITELN(RES_OUT,LINE);
      END;
    END;
    CLOSE(SORT_IN);
  END;

  WRITELN(RES_OUT,'/END_EDGES');

 {VERTICES DATA : }

  WRITELN('SAVING VERTICES DATA :: POCKET ',I,FACE_TYPE);

  WRITELN(RES_OUT,'/VERTICES');

  SORT_FILE := CONCAT(PART_NAME,'.VTX');
  SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

  ASSIGN(SORT_IN,SORT_FILE);

  FOR II := 1 TO RES_VERT_CNT DO
  BEGIN
    RESET(SORT_IN);
    FOR J := 1 TO NUMVERTS DO
    BEGIN
      READLN(SORT_IN,LINE);
      ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
      VAL(ID_NUM_STR,ID_NUM,CODE);
      IF (ID_NUM = RES_VERT_NUM[II]) THEN
      BEGIN
        WRITELN(RES_OUT,LINE);
      END;
    END;
    CLOSE(SORT_IN);
  END;

  WRITELN(RES_OUT,'/END_VERTICES');
```

```
{SURFACE DATA :}

 WRITELN('SAVING SURFACE DATA :: POCKET ',I,FACE_TYPE);
 WRITELN(RES_OUT,'/END_TOPOLOGY');
 WRITELN(RES_OUT,'/GEOMETERY');

 WRITELN(RES_OUT,'/SURFACES');

 SORT_FILE := CONCAT(PART_NAME,'.SUR');
 SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

 ASSIGN(SORT_IN,SORT_FILE);
 RESET(SORT_IN);

 FOR II := 1 TO NUMSURFS DO
 BEGIN
   READLN(SORT_IN,LINE);
   ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
   VAL(ID_NUM_STR,ID_NUM,CODE);
   IF (ID_NUM = RES_SURF) THEN
   BEGIN
     WRITELN(RES_OUT,LINE);
   END;
 END;
 WRITELN(RES_OUT,'/END_SURFACES');
 CLOSE(SORT_IN);

{CURVE_DATA}

 WRITELN('SAVING CURVE DATA :: POCKET ',I,FACE_TYPE);

 WRITELN(RES_OUT,'/CURVES');

 SORT_FILE := CONCAT(PART_NAME,'.CRV');
 SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

 ASSIGN(SORT_IN,SORT_FILE);

 FOR II := 1 TO RES_CURVE_CNT DO
 BEGIN
   RESET(SORT_IN);
   FOR J := 1 TO NUMCURVS DO
   BEGIN
     READLN(SORT_IN,LINE);
```

```
      ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
      VAL(ID_NUM_STR,ID_NUM,CODE);
      IF (ID_NUM = RES_CURVE_NUM[II]) THEN
      BEGIN
        WRITELN(RES_OUT,LINE);
      END;
    END;
    CLOSE(SORT_IN);
  END;

  WRITELN(RES_OUT,'/END_CURVES');

 {POINTS DATA}

  WRITELN('SAVING POINT DATA :: POCKET ',I,FACE_TYPE);
  WRITELN(RES_OUT,'/POINTS');

  SORT_FILE := CONCAT(PART_NAME,'.PNT');
  SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

  ASSIGN(SORT_IN,SORT_FILE);

  FOR II := 1 TO RES_POINT_CNT DO
  BEGIN
    RESET(SORT_IN);
    FOR J := 1 TO NUMPOINTS DO
    BEGIN
      READLN(SORT_IN,LINE);
      ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
      VAL(ID_NUM_STR,ID_NUM,CODE);
      IF (ID_NUM = RES_POINT_NUM[II]) THEN
      BEGIN
        WRITELN(RES_OUT,LINE);
      END;
    END;
    CLOSE(SORT_IN);
  END;

  WRITELN(RES_OUT,'/END_POINTS');

 {VECTOR DATA :}

  WRITELN('SAVING VECTOR DATA :: POCKET ',I,FACE_TYPE);

  WRITELN(RES_OUT,'/UNIT_VECTORS');
```

```
    SORT_FILE := CONCAT(PART_NAME,'.VEC');
    SORT_FILE := CONCAT('C:\TP\SOURCE\',SORT_FILE);

    ASSIGN(SORT_IN,SORT_FILE);
    RESET(SORT_IN);

    FOR II := 1 TO RES_VEC_CNT DO
    BEGIN
      RESET(SORT_IN);
      FOR J := 1 TO NUMVECS DO
      BEGIN
        READLN(SORT_IN,LINE);
        ID_NUM_STR := CONCAT(LINE[6],LINE[7],LINE[8]);
        VAL(ID_NUM_STR,ID_NUM,CODE);
        IF (ID_NUM = RES_VEC_NUM[II]) THEN
        BEGIN
          WRITELN(RES_OUT,LINE);
        END;
      END;
      CLOSE(SORT_IN);
    END;

    WRITELN(RES_OUT,'/END_UNIT_VECTORS');
    WRITELN(RES_OUT,'/END_GEOMETRY');

    IF (FACE_TYPE = 'S') THEN
    BEGIN
      WRITELN(RES_OUT,'/END_SIDE_PLANE');
      CLOSE(RES_OUT);
    END;


    IF (FACE_TYPE = 'B') THEN
    BEGIN
      WRITELN(RES_OUT,'/END_BOTTOM_PLANE');
      CLOSE(RES_OUT);
    END;

END; {store_poc_data}

{ ==========================================================}

PROCEDURE FIND_POCKETS;
```

Appendix L (FEATURE_ID Program Code Unit POCKET)

```
{This routine performs the analysis of the part model data
in order to find any pocket removal features.

Parent: FIND_POCKETS

Children: CONSOLODATE, READ_DATA, STORE_POC_DATA, TEXTCOLOR}

{ -----------------------------------------------------------}

BEGIN

   PART_NAME := PART_ANAL_ID;
   READ_DATA;

   TEXTCOLOR(12);

   WRITELN;
   WRITELN('ANALYSING ',PART_ANAL_ID,':: POCKETS');
   WRITELN;

   RES_FACE_CNT := 0;
   RES_LOOP_CNT := 0;
   RES_EDGE_CNT := 0;
   RES_VERT_CNT := 0;
   RES_SURF_CNT := 0;
   RES_CURVE_CNT := 0;
   RES_POINT_CNT := 0;
   RES_VEC_CNT := 0;

{FIND FEATURE ELEMENTS FOR POCKET}

 {FIND Z PLANE UNIT VECTOR}

   FOR I := 1 TO NUMVECS DO
   BEGIN
     IF (VECTORS[I].UX1 = 0) AND (VECTORS[I].
     UY1 = 0) AND (VECTORS[I].UZ1 = 1) THEN
     BEGIN
        RES_VEC := I;
     END; {IF}
   END; {FOR}

      { GET SURFACE }

   RES_SURF := 0;
```

```
FOR I := 1 TO NUMSURFS DO
BEGIN
  IF (SURFACES[I].SURF_VEC_NUM = RES_VEC) THEN
  BEGIN
    IF (SURFACES[I].SURF_DELTA > RES_SURF) THEN
    BEGIN
       RES_SURF := I;
    END; {IF}
  END; {IF}
END; {FOR}

{ GET DATA }

A := 1;

FOR I := 1 TO NUMFACES DO          {FACES}
BEGIN
  IF (FACES[I].FAC_SURF_NUM = RES_SURF) THEN
  BEGIN
    RES_FACE_NUM[A] := I;
    RES_FACE_CNT := RES_FACE_CNT + 1;
    A := A + 1;
  END;
END;

A := 1;
FOR I := 1 TO RES_FACE_CNT DO     {CHECK FOR POCKETS}
BEGIN
  FOR J := 2 TO 25 DO
  BEGIN
    IF FACES[RES_FACE_NUM[I]].FAC_LOOP_NUMS[J] > 0 THEN
    BEGIN
      RES_LOOP_NUM[A] :=
          FACES[RES_FACE_NUM[I]].FAC_LOOP_NUMS[J];
      POCKET_FOUND := TRUE;
      A := A + 1;

      {get loop id for a pocket)

      RES_LOOP_CNT := RES_LOOP_CNT + 1;
    END;
  END;
END;

IF POCKET_FOUND THEN
```

Appendix L (FEATURE_ID Program Code Unit POCKET)

```
BEGIN
  A := 1;
  J := 1;
  NUM_TEMP_EDGES := 0;
  FEAT_NUMBER := 0;

  {FOR EACH POCKET FOUND}

  FOR I := 1 TO RES_LOOP_CNT DO

  {GET EDGES FOR OPENING}

  BEGIN
    FOR J := 1 TO 25  DO
    BEGIN
    IF (LOOPS[RES_LOOP_NUM[I]].LOOP_EDG_NUMS[J]>0)
       THEN
      BEGIN
        TEMP_EDGE[A] :=
            LOOPS[RES_LOOP_NUM[I]].LOOP_EDG_NUMS[J];
        A := A + 1;
        NUM_TEMP_EDGES := NUM_TEMP_EDGES + 1;
      END;
    END;

    B := 1;
    NUM_TEMP_SIDE_LOOP := 0;

    {GET LOOPS FOR POCKET SIDES}

    FOR C := 1 TO NUMLOOPS DO
    BEGIN
      FOR D := 1 TO NUM_TEMP_EDGES DO
      BEGIN
        FOR E := 1 TO 25 DO
        BEGIN
          IF C <> RES_LOOP_NUM[I] THEN
          BEGIN
          IF (LOOPS[C].LOOP_EDG_NUMS[E] =
              TEMP_EDGE[D]) THEN
            BEGIN
              TEMP_SIDE_LOOP[B] := C;
              B := B + 1;
              NUM_TEMP_SIDE_LOOP :=
                 NUM_TEMP_SIDE_LOOP + 1;
```

```
          END;
        END;
      END;
    END;
  END;                      {WE HAVE LOOP NUMS FOR SIDES}
                            {NUMBER OF SIDES = N_T_S_L}

  {GET LOOP FOR POCKET BOTTOM}

  {LOAD UP ALL SIDE EDGE NUMS INTO B_C}

  F := 1;
  NUM_CHECK_EDGES := 0;

  FOR G := 1 TO NUM_TEMP_SIDE_LOOP DO
  BEGIN
    FOR H := 1 TO 25 DO
    BEGIN
      IF (LOOPS[TEMP_SIDE_LOOP[G]]
          .LOOP_EDG_NUMS[H] > 0) THEN
      BEGIN
        BOTTOM_CHECK[F] :=
         LOOPS[TEMP_SIDE_LOOP[G]].LOOP_EDG_NUMS[H];
        F := F + 1;
        NUM_CHECK_EDGES := NUM_CHECK_EDGES + 1;
      END;
    END;
  END;

  FOR AA := 1 TO 25 DO
  {LOAD UP ALL EDGES FROM OPENING LOOP}
  BEGIN
    IF (LOOPS[RES_LOOP_NUM[I]]
        .LOOP_EDG_NUMS[AA] > 0) THEN
    BEGIN
      BOTTOM_CHECK[F] := LOOPS
           [RES_LOOP_NUM[I]].LOOP_EDG_NUMS[AA];
      F := F + 1;
      NUM_CHECK_EDGES := NUM_CHECK_EDGES + 1;
    END;
  END;

  {CLEAR ANY DUPLICATES FROM BOTTOM_CHECK}
  {THIS WILL LEAVE THE EDGE NUMS FOR THE POCKET BOTTOM}
```

```
FOR BB := 1 TO NUM_CHECK_EDGES DO
BEGIN
  TEMP := BOTTOM_CHECK[BB];
  FOR CC := BB+1 TO NUM_CHECK_EDGES DO
  BEGIN
    IF (BOTTOM_CHECK[CC] = TEMP) THEN
    BEGIN
      BOTTOM_CHECK[CC] := -111;
      BOTTOM_CHECK[BB] := -111;
    END;
  END;
END;

{SEARCH FOR LOOP CONTAINING REMAINING EDGES}

FOR DD := 1 TO NUMLOOPS DO
BEGIN
  CHECK_COUNT := 0;
  FOR EE := 1 TO 25 DO
  BEGIN
    FOR FF := 1 TO NUM_CHECK_EDGES DO
    BEGIN
      IF (LOOPS[DD].LOOP_EDG_NUMS[EE] =
      BOTTOM_CHECK[FF]) THEN
      BEGIN
         CHECK_COUNT := CHECK_COUNT + 1;
      END;
    END;
  END;
  IF (CHECK_COUNT > 1) THEN
    {IF MORE THAN 1 EDGE MATCHES)
  BEGIN
    BOTTOM_LOOP := DD;
  END;
END;

{GET FACES ASSOCIATED WITH SIDE/BOTTOM LOOPS}

A := 1;
SIDE_FACE_CNT := 0;

FOR GG := 1 TO NUMFACES DO        {SIDE FACES}
BEGIN
  FOR HH := 1 TO NUM_TEMP_SIDE_LOOP DO
  BEGIN
```

```
      IF (TEMP_SIDE_LOOP[HH] =
        FACES[GG].FAC_LOOP_NUMS[1]) THEN
      BEGIN
        SIDE_FACE[A] := GG;
        A := A + 1;
        SIDE_FACE_CNT := SIDE_FACE_CNT + 1;
      END;
    END;
  END;

  FOR GG := 1 TO NUMFACES DO          {BOTTOM FACE}
  BEGIN
    IF (BOTTOM_LOOP = FACES[GG]
       .FAC_LOOP_NUMS[1]) THEN
    BEGIN
      BOTTOM_FACE := GG;
    END;
  END;

  { CONSOLIDATE DATA FOR EACH FACE FOR OUTPUT TO }
  { FEATURE FILE }

  {SIDE FACES}

  FOR GG := 1 TO SIDE_FACE_CNT DO
  BEGIN
    STR(I,FACE_NUM);
    FACE_TYPE := 'S';
    FEAT_NUMBER := FEAT_NUMBER + 1;
    STR(FEAT_NUMBER,FEAT_NUM);

    FACE_NUMBER := SIDE_FACE[GG];

    CONSOLIDATE;

    STORE_POC_DATA;

  END;

  {FOR BOTTOM}

  FACE_TYPE := 'B';

  FEAT_NUMBER := FEAT_NUMBER + 1;
```

```
       STR(FEAT_NUMBER,FEAT_NUM);

       FACE_NUMBER := BOTTOM_FACE;

       CONSOLIDATE;

       STORE_POC_DATA;

     END;  { FOR EACH POCKET FOUND }

   END; { IF POCKET FOUND }

   IF (NOT POCKET_FOUND) THEN
   BEGIN
     WRITELN;
     WRITELN('NO POCKET FOUND');
     WRITELN;
   END;

END;  {find_pockets}


END.  {unit pocket}
```

Appendix M (FEATURE_ID Program Code Unit Arrays)

```
UNIT ARRAYS;

{Unit ARRAYS containd the data dictionary reading routines
which lexically scan the input databse files to load design
geometric data into the program.}

{ *********************************************************}

INTERFACE

USES {units}

  CRT,
  COMMON;


{subroutines with global access}

PROCEDURE READ_VECTORS;
PROCEDURE READ_POINTS;
PROCEDURE READ_CURVES;
PROCEDURE READ_SURFACES;
PROCEDURE READ_VERTICES;
PROCEDURE READ_EDGES;
PROCEDURE READ_LOOPS;
PROCEDURE READ_FACES;
PROCEDURE READ_SHELLS;

{----------------------------------------------------------}

IMPLEMENTATION

{local unit variables}

TYPE

  LINE = STRING[100];           {line type string}

VAR

  R_LINE       : LINE;          {line read}
  TEMP_LINE    : LINE;          {line to hold number char.}
  LAST_LINE    : LINE;          {previously read line}
  CODE         : INTEGER;       {error code}
  MARK         : INTEGER;       {loop counter flag}
```

```
  I,J,K        : INTEGER;       {cunters}
{=============================================================}
PROCEDURE READ_VECTORS;

{ This routine reads in vector data.

Parent: READ_DATA

Children: VAL     }

{ -----------------------------------------------------------}

{local file id variables}

VAR

  VEC_IN      : TEXT;
  VECTOR_FILE : STRING[50];


BEGIN

  VECTOR_FILE := CONCAT(PART_DIR,'.VEC');
  ASSIGN(VEC_IN,VECTOR_FILE);
  RESET(VEC_IN);

  ARRAY_LOC := 0;
  NUMVECS := 0;

  WHILE NOT EOF(VEC_IN) DO
  BEGIN
    READLN(VEC_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
    BEGIN
      IF (R_LINE[I]='V') AND
         (R_LINE[I+1]='E')AND(R_LINE[I+2]='C') THEN
      BEGIN
        ARRAY_LOC := ARRAY_LOC + 1;
      END;
      TEMP_LINE := '9999999';
      IF (R_LINE[I]=';') AND (R_LINE[I+2] IN ['0'..'9'])
```

```
     THEN
    BEGIN
      FOR J := I+2 TO I+8 DO
      BEGIN
        TEMP_LINE[J-2-I+1] := R_LINE[J];
  END;
      VAL(TEMP_LINE,VECTORS[ARRAY_LOC].UX1,CODE);
      IF CODE <> 0 THEN
      BEGIN
        WRITELN('PROBLEM WITH VECTOR UX1 TRANSLATION');
      END;
      MARK := MARK+1;
    END;
    IF (R_LINE[I]=',') AND (R_LINE[I+2] IN
        ['0'..'9']) AND (MARK=2) THEN
    BEGIN
      FOR J := I+2 TO I+8 DO
      BEGIN
        TEMP_LINE[J-2-I+1] := R_LINE[J];
      END;
      VAL(TEMP_LINE,VECTORS[ARRAY_LOC].UY1,CODE);
      IF CODE <> 0 THEN
      BEGIN
        WRITELN('PROBLEM WITH VECTOR UY1 TRANSLATION');
      END;
      MARK := MARK+1;
      I := I + 1;
    END;

    IF (R_LINE[I]=',') AND (R_LINE[I+2] IN
        ['0'..'9']) AND (MARK=3) THEN
    BEGIN
      FOR J := I+2 TO I+8 DO
      BEGIN
        TEMP_LINE[J-2-I+1] := R_LINE[J];
      END;
      VAL(TEMP_LINE,VECTORS[ARRAY_LOC].UZ1,CODE);
      IF CODE <> 0 THEN
      BEGIN
        WRITELN('PROBLEM WITH VECTOR UY1 TRANSLATION');
      END;
      MARK := MARK+1;
    END;
    I := I + 1
  END;
```

```
    NUMVECS := NUMVECS + 1;
  END;

  CLOSE(VEC_IN);

END; {read_vectors}

{===========================================================}

PROCEDURE READ_POINTS;

{This routine reads point data into the program.

 Parent: READ_DATA

 CHildren: VAL   }

{ ---------------------------------------------------------}

{local file id vars.}

VAR

  PNT_IN      : TEXT;
  POINT_FILE : STRING[50];

BEGIN

  POINT_FILE := CONCAT(PART_DIR,'.PNT');
  ASSIGN(PNT_IN,POINT_FILE);
  RESET(PNT_IN);
  POINT_LN_NUM := 1;
  ARRAY_LOC := 0;
  NUMPOINTS := 0;
  WHILE NOT EOF(PNT_IN) DO
  BEGIN
    READLN(PNT_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
    BEGIN
       IF (R_LINE[I]='P') AND (R_LINE[I+1]='N')
          AND(R_LINE[I+2]='T') THEN
   BEGIN
        TEMP_LINE := CONCAT
```

```
        (R_LINE[I+3],R_LINE[I+4],R_LINE[I+5]);
        VAL(TEMP_LINE,POINT_ID,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH PNTNUM TRANSLATION');
        END;

        POINT_LN[POINT_ID] := POINT_LN_NUM;
        POINT_LN_NUM := POINT_LN_NUM + 1;
        ARRAY_LOC := ARRAY_LOC + 1;

      END;

      TEMP_LINE := '9999999';

      IF (R_LINE[I]=';') AND (R_LINE[I+2] IN ['0'..'9'])
      THEN
      BEGIN
        FOR J := I+2 TO I+8 DO
        BEGIN
          TEMP_LINE[J-2-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,POINTS[ARRAY_LOC].X1,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH POINT X1 TRANSLATION');
        END;
        MARK := MARK+1;
      END;

      IF (R_LINE[I]=',') AND (R_LINE[I+2] IN ['0'..'9']) AND
      (MARK=2) THEN
      BEGIN
        FOR J := I+2 TO I+8 DO
        BEGIN
          TEMP_LINE[J-2-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,POINTS[ARRAY_LOC].Y1,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH POINT Y1 TRANSLATION');
        END;
        MARK := MARK+1;
        I := I + 1;
      END;
```

```
      IF (R_LINE[I]=',') AND (R_LINE[I+2] IN ['0'..'9']) AND
         (MARK=3) THEN
      BEGIN
        FOR J := I+2 TO I+8 DO
        BEGIN
          TEMP_LINE[J-2-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,POINTS[ARRAY_LOC].Z1,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH POINTS Y1 TRANSLATION');
        END;
        MARK := MARK+1;
      END;

    I = I + 1;
    END;

  NUMPOINTS := NUMPOINTS + 1;

  END;
  CLOSE(PNT_IN);

END; {read_points}

{==========================================================}

PROCEDURE READ_CURVES;

{ Thid routine reads curve data into the program.

Parent: READ_DATA

Children: VAL   }

{ -----------------------------------------------------------}

{local file id vars.}

VAR

  CRV_IN      : TEXT;
  CURVE_FILE : STRING[50];

BEGIN
```

```
CURVE_FILE := CONCAT(PART_DIR,'.CRV');
ASSIGN(CRV_IN,CURVE_FILE);
RESET(CRV_IN);
ARRAY_LOC := 0;
NUMCURVS := 0;

WHILE NOT EOF(CRV_IN) DO
BEGIN
  READLN(CRV_IN,R_LINE);
  I := 1;
  MARK := 1;
  WHILE I <> LENGTH(R_LINE) DO
  BEGIN
    IF (R_LINE[I]='C') AND
    (R_LINE[I+1]='R')AND(R_LINE[I+2]='V') THEN
    BEGIN
      ARRAY_LOC := ARRAY_LOC + 1;
    END;

    IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'L')THEN
    BEGIN
      CURVES[ARRAY_LOC].CRV_TYPE := 'LINE';
      I := I + 1;
    END;

    IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'C')THEN
    BEGIN
      CURVES[ARRAY_LOC].CRV_TYPE := 'CIRC';
      I := I + 1;
    END;

    TEMP_LINE := '999';

    IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'P') THEN
    BEGIN
      FOR J := I+4 TO I+6 DO
      BEGIN
        TEMP_LINE[J-4-I+1] := R_LINE[J];
      END;

      VAL(TEMP_LINE,CURVES[ARRAY_LOC].CRV_PNT_NUM,CODE);

      IF CODE <> 0 THEN
      BEGIN
```

```
          WRITELN('PROBLEM WITH CURVES POINT_NUM
                   TRANSLATION');
        END;
        I := I + 1;
      END;

      IF (R_LINE[I]=',') AND (R_LINE[I+2] = 'V') THEN
      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;

        VAL(TEMP_LINE,CURVES[ARRAY_LOC].CRV_VEC_NUM,CODE);

        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH CURVES VECTOR_NUM
                   TRANSLATION');
        END;
        I := I + 1;
      END;
      I := I + 1
    END;

  NUMCURVS := NUMCURVS + 1;

  END;

 CLOSE(CRV_IN);

END; {read_curves}

 { ==================================================}

PROCEDURE READ_SURFACES;

{ This routine reads surface data into the program.

Parent: READ_DATA

Children: VAL  }

{ ----------------------------------------------------}
```

Appendix M (FEATURE_ID Program Code Unit Arrays)

```
{local file_id vars.}

VAR

  SUR_IN      : TEXT;
  SURF_FILE : STRING[50];

BEGIN

  SURF_FILE := CONCAT(PART_DIR,'.SUR');
  ASSIGN(SUR_IN,SURF_FILE);
  RESET(SUR_IN);

  ARRAY_LOC := 0;
  NUMSURFS := 0;

  WHILE NOT EOF(SUR_IN) DO
  BEGIN
    READLN(SUR_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
    BEGIN
      IF (R_LINE[I]='S') AND
         (R_LINE[I+1]='U')AND(R_LINE[I+2]='R') THEN
      BEGIN
        ARRAY_LOC := ARRAY_LOC + 1;
      END;

      IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'P')THEN
      BEGIN
        SURFACES[ARRAY_LOC].SURF_TYPE := 'PLANE';
        I := I + 1;
      END;

      TEMP_LINE := '999';

      IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'V') THEN
      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,SURFACES[ARRAY_LOC]
           .SURF_VEC_NUM,CODE);
```

```
          IF CODE <> 0 THEN
          BEGIN

            WRITELN('PROBLEM WITH SURF VEC_NUM TRANSLATION');
          END;
          I := I + 1;
        END;

        TEMP_LINE := '9999999';

        IF (R_LINE[I]=';') AND (R_LINE[I+2] IN ['0'..'9'])
           THEN
        BEGIN
          FOR J := I+2 TO I+8 DO
          BEGIN
            TEMP_LINE[J-2-I+1] := R_LINE[J];
          END;
          VAL(TEMP_LINE,SURFACES[ARRAY_LOC].SURF_DELTA,CODE);
          IF CODE <> 0 THEN
          BEGIN
            WRITELN('PROBLEM WITH SURF_DELTA TRANSLATION');
          END;
        END;
        I := I + 1
      END;

    NUMSURFS := NUMSURFS + 1;

    END;

  CLOSE(SUR_IN);

END;  {read_surfaces}

{============================================================}

PROCEDURE READ_VERTICES;

{ This routine reads vertices data into the progam.

Parent: READ_DATA

Children: VAL    }

{ ----------------------------------------------------------}
```

Appendix M (FEATURE_ID Program Code Unit Arrays)

```
(local file_id vars.)

VAR

  VTX_IN       : TEXT;
  VERT_FILE : STRING[50];

BEGIN
  VERT_FILE := CONCAT(PART_DIR,'.VTX');
  ASSIGN(VTX_IN,VERT_FILE);
  RESET(VTX_IN);

  ARRAY_LOC := 0;
  NUMVERTS := 0;

  WHILE NOT EOF(VTX_IN) DO
  BEGIN
    READLN(VTX_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
    BEGIN
      IF (R_LINE[I]='V') AND
          (R_LINE[I+1]='T')AND(R_LINE[I+2]='X') THEN
      BEGIN
        ARRAY_LOC := ARRAY_LOC + 1;
      END;

      TEMP_LINE := '999';

      IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'P') THEN

      BEGIN
        FOR J := I+4 TO I+6 DO
        BEGIN
          TEMP_LINE[J-4-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,VERTICEES[ARRAY_LOC].
            VERT_PNT_NUM,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH VERTEX
                   POINT_NUM TRANSLATION');
        END;
        I := I + 1;
```

```
      END;
      I := I + 1
    END;

  NUMVERTS := NUMVERTS + 1;

  END;
  CLOSE(VTX_IN);

END; {read_vertices}

{==========================================================}

PROCEDURE READ_EDGES;

{ This routine read edge data into the program.

Parent: READ_DATA

Children: VAL    }

{ --------------------------------------------------------}

{local file_id vars.}

VAR

  EDG_IN      : TEXT;
  EDGE_FILE : STRING[50];

BEGIN

  EDGE_FILE := CONCAT(PART_DIR,'.EDG');
  ASSIGN(EDG_IN,EDGE_FILE);
  RESET(EDG_IN);

  ARRAY_LOC := 0;
  NUMEDGES := 0;

  WHILE NOT EOF(EDG_IN) DO
  BEGIN
    READLN(EDG_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
```

```
    BEGIN
      IF (R_LINE[I]='E') AND
          (R_LINE[I+1]='D')AND(R_LINE[I+2]='G') THEN
      BEGIN
        ARRAY_LOC := ARRAY_LOC + 1;
      END;

      TEMP_LINE := '999';

      IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'V') THEN

      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;

        VAL(TEMP_LINE,EDGES[ARRAY_LOC].EDG_VERT_NUM_1,CODE);

        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH EDGE_VERTEX POINT_N_1
                   TRANSLATION');
        END;
        I := I + 1;
      END;

      IF (R_LINE[I]=',') AND (R_LINE[I+2] = 'V') THEN
      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,EDGES[ARRAY_LOC].EDG_VERT_NUM_2,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH EDGE VERTEX POINT_N_2
                   TRANSLATION');
        END;
        I := I + 1;
      END;
      IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'C') THEN
      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
```

```
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,EDGES[ARRAY_LOC].EDG_CRV_NUM,CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM WITH EDGE_CRV_NUM TRANSLATION');
        END;
        I := I + 1;
      END;

      IF (R_LINE[I] = '+') THEN
      BEGIN
        EDGES[ARRAY_LOC].EDG_SENSE := 1;
      END;
      IF (R_LINE[I] = '-') THEN
      BEGIN
        EDGES[ARRAY_LOC].EDG_SENSE := -1;
      END;
      I := I + 1
    END;

  NUMEDGES := NUMEDGES + 1;
  END;

CLOSE(EDG_IN);

END;  {read_edges}

{===========================================================}

PROCEDURE READ_LOOPS;

{This routine reads loop data into the program.

Parent: READ_DATA

Children: VAL    }

{ ---------------------------------------------------------}
{local file_id vars.}

VAR

  LOP_IN     : TEXT;
  LOOP_FILE : STRING[50];
```

Appendix M (FEATURE_ID Program Code Unit Arrays)

```
BEGIN

  FOR I := 1 TO 50 DO
  BEGIN
    FOR J := 1 TO 25 DO
    BEGIN
      LOOPS[I].LOOP_EDG_NUMS[J] := 0;
    END;
  END;

  LOOP_FILE := CONCAT(PART_DIR,'.LOP');
  ASSIGN(LOP_IN,LOOP_FILE);
  RESET(LOP_IN);

  ARRAY_LOC := 0;
  NUMLOOPS := 0;
  K := 1;

  WHILE NOT EOF(LOP_IN) DO
  BEGIN
    READLN(LOP_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
    BEGIN
      IF (R_LINE[I]='L') AND
          (R_LINE[I+1]='O')AND(R_LINE[I+2]='P') THEN
      BEGIN
        ARRAY_LOC := ARRAY_LOC + 1;
        MARK := 1;
        K := 1;
      END;

      TEMP_LINE := '999';

      IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'E') THEN
      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;

        VAL(TEMP_LINE,LOOPS[ARRAY_LOC].
            LOOP_EDG_NUMS[K],CODE);
        IF CODE <> 0 THEN
```

```
        BEGIN
          WRITELN('PROBLEM WITH LOOP_EDG_NUMS TRANSLATION');
        END;
        K := K + 1;
        I := I + 1;
      END;

      IF (R_LINE[I]=',') AND (R_LINE[I+2] = 'E') THEN
      BEGIN
        FOR J := I+5 TO I+7 DO
        BEGIN
          TEMP_LINE[J-5-I+1] := R_LINE[J];
        END;
        VAL(TEMP_LINE,LOOPS[ARRAY_LOC]
           .LOOP_EDG_NUMS[K],CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM  WITH LOOP_EDGE_2 TRANSLATION');
        END;
        I := I + 1;
        K := K + 1;
      END;
      IF (R_LINE[I] = '+') THEN
      BEGIN
        LOOPS[ARRAY_LOC].LOOP_SENSE[K-1] := 1;
      END;
      IF (R_LINE[I] = '-') THEN
      BEGIN
        LOOPS[ARRAY_LOC].LOOP_SENSE[K-1] := -1;
      END;
      I := I + 1
    END;
  END;
  NUMLOOPS := NUMLOOPS + 1;
  END;
  CLOSE(LOP_IN);

END; {READ_LOOPS}

{=========================================================}

PROCEDURE READ_FACES;

{ This routine reads face data into the program.
```

```
Appendix M  (FEATURE_ID Program Code Unit Arrays)

Parent: READ_DATA

Children: VAL    }

{ -----------------------------------------------------------}
{ local file_id vars.}

VAR

  FAC_IN      : TEXT;
  FACE_FILE : STRING[50];


BEGIN

  FOR I := 1 TO 50 DO
  BEGIN
    FOR J := 1 TO 25 DO
    BEGIN
      FACES[I].FAC_LOOP_NUMS[J] := 0;
    END;
  END;

  FACE_FILE := CONCAT(PART_DIR,'.FAC');
  ASSIGN(FAC_IN,FACE_FILE);
  RESET(FAC_IN);

  ARRAY_LOC := 0;
  NUMFACES := 0;
  K := 1;

  WHILE NOT EOF(FAC_IN) DO
  BEGIN
    READLN(FAC_IN,R_LINE);
    I := 1;
    MARK := 1;
    WHILE I <> LENGTH(R_LINE) DO
    BEGIN
      IF (R_LINE[I]='F') AND
          (R_LINE[I+1]='A')AND(R_LINE[I+2]='C') THEN
      BEGIN
        ARRAY_LOC := ARRAY_LOC + 1;
        MARK := 1;
        K := 1;
      END;
```

```
TEMP_LINE := '999';

IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'L') THEN
BEGIN
  FOR J := I+5 TO I+7 DO
  BEGIN
    TEMP_LINE[J-5-I+1] := R_LINE[J];
  END;
  VAL(TEMP_LINE,FACES[ARRAY_LOC]
      .FAC_LOOP_NUMS[K],CODE);
  IF CODE <> 0 THEN
  BEGIN
    WRITELN('PROBLEM WITH FACE_LOOP_NUMS
             TRANSLATION');
  END;
  K := K + 1;
  I := I + 1;
END;

IF (R_LINE[I]=',') AND (R_LINE[I+2] = 'L') THEN
BEGIN
  FOR J := I+5 TO I+7 DO
  BEGIN
    TEMP_LINE[J-5-I+1] := R_LINE[J];
  END;
  VAL(TEMP_LINE,FACES[ARRAY_LOC]
      .FAC_LOOP_NUMS[K],CODE);
  IF CODE <> 0 THEN
  BEGIN
    WRITELN('PROBLEM  WITH FACE_LOOP_2 TRANSLATION');
  END;
  I := I + 1;
  K := K + 1;
END;

IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'S') THEN
BEGIN
  FOR J := I+5 TO I+7 DO
  BEGIN
    TEMP_LINE[J-5-I+1] := R_LINE[J];
  END;
  VAL(TEMP_LINE,FACES[ARRAY_LOC].FAC_SURF_NUM,CODE);
  IF CODE <> 0 THEN
  BEGIN
    WRITELN('PROBLEM WITH FACE_LOOP_NUMS
```

```
                    TRANSLATION');
        END;
        I := I + 1;
      END;

      IF (R_LINE[I] = '+') THEN
      BEGIN
        FACES[ARRAY_LOC].FAC_SENSE := 1;
      END;

      IF (R_LINE[I] = '-') THEN
      BEGIN
        FACES[ARRAY_LOC].FAC_SENSE := -1;
      END;
      I := I + 1
    END;
  NUMFACES := NUMFACES + 1;

  END;
  CLOSE(FAC_IN);

END;  {read_faces}

{=============================================================}

PROCEDURE READ_SHELLS;

{ This routine reads shell data into the program.

Parent : READ_DATA

Children: VAL   }

{ -----------------------------------------------------------}
{local file_id vars.}

VAR

  SHL_IN     : TEXT;
  SHEL_FILE : STRING[50];

BEGIN

  SHEL_FILE := CONCAT(PART_DIR,'.SHL');
  ASSIGN(SHL_IN,SHEL_FILE);
```

```
RESET(SHL_IN);
ARRAY_LOC := 0;
NUMSHELLS := 0;
K := 1;

WHILE NOT EOF(SHL_IN) DO
BEGIN
  READLN(SHL_IN,R_LINE);
  I := 1;
  MARK := 1;
  WHILE I <> LENGTH(R_LINE) DO
  BEGIN
    IF (R_LINE[I]='S') AND
       (R_LINE[I+1]='H')AND(R_LINE[I+2]='L') THEN
    BEGIN
      ARRAY_LOC := ARRAY_LOC + 1;
      MARK := 1;
      K := 1;
    END;

    TEMP_LINE := '999';

    IF (R_LINE[I]=';') AND (R_LINE[I+2] = 'F') THEN
    BEGIN
      FOR J := I+5 TO I+7 DO
      BEGIN
        TEMP_LINE[J-5-I+1] := R_LINE[J];
      END;
      VAL(TEMP_LINE,SHELLS[ARRAY_LOC]
          .SHL_FAC_NUM[K],CODE);
      IF CODE <> 0 THEN
      BEGIN
        WRITELN('PROBLEM WITH SHELL_FACE_NUM
                 TRANSLATION');
      END;
      K := K + 1;
      I := I + 1;
    END;

    IF (R_LINE[I]=',') AND (R_LINE[I+2] = 'F') THEN
    BEGIN
      FOR J := I+5 TO I+7 DO
      BEGIN
        TEMP_LINE[J-5-I+1] := R_LINE[J];
      END;
```

```
        VAL(TEMP_LINE,SHELLS[ARRAY_LOC].SHL_FAC_NUM[K],CODE);
        IF CODE <> 0 THEN
        BEGIN
          WRITELN('PROBLEM  WITH FACE_LOOP_2 TRANSLATION');
        END;
        I := I + 1;
        K := K + 1;
      END;

      I := I + 1;
    END;

  NUMSHELLS := NUMSHELLS + 1;
  END;

  CLOSE(SHL_IN);

END; {read_shells}

END. {unit arrays}
```

Appendix N (Pseudo Code for TOP_FACE Identification)

To interpret and define a TOP_FACE feature-of-removal from given part and part blank input BREP databases, the FEATURE_ID program uses the following logic.

1. Find the ENTRY PLANE feature component of the TOP_FACE feature [Figure N].

    - Read part blank BREP database into program.
    - Scan vector definitions.
        - If Z component of a vector > 0 then save vector ID #.
    - Scan surface definitions.
        - If vector of a surface = vector ID from above then save surface ID #.
    - Find ENTRY PLANE
        - Scan face definitions
            - If surface of face is from above then save face ID #.
        - For each face found
            - determine total sum of z coordinates of all vertices associated with the face.
        - ENTRY PLANE is the face with the largest sum of z coordinates.
    - Store data associated with ENTRY PLANE component.
        - Pull all BREP definitions associated with ENTRY PLANE and place in output database.

2. Find the CHECK PLANE feature component of the TOP_FACE feature [Figure N].

    - Read part BREP database into program.
    - Scan vector definitions.
        - If Z component of a vector > 0 then save vector ID #.
    - Scan surface definitions.
        - If vector of a surface = vector ID from above then save surface ID #.
    - Find CHECK PLANE
        - Scan face definitions
            - If surface of face is from above then save

```
                face ID #.
            - For each face found
                - determine total sum of z coordinates of
                  all vertices associated with the face.
            - CHECK PLANE is the face with the largest sum
              of z coordinates.
        - Store data associated with CHECK PLANE component.
            - Pull all BREP definitions associated with
              CHECK PLANE and place in output database.
```

The result of the TOP_FACE analysis provides the definition of the TOP_FACE feature associated with a given part and part blank in terms of two components. The first component or ENTRY PLANE represents the surface were material will first be experienced by the NC machine tool. The second component or CHECK PLANE represents the surface were the finished part material is located, and therefore, the bottom of the TOP_FACE feature.

APPENDIX N (TOP_FACE FEATURE IDENTIFICATION)

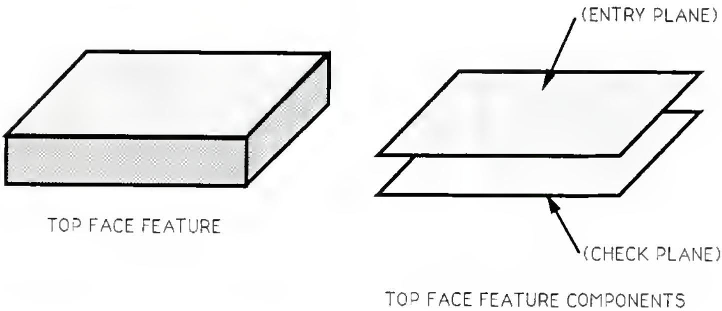


FIGURE N - Components of the TOP_FACE feature.

Appendix O (Pseudo Code for POCKET Identification)

To interpret and define a POCKET feature-of-removal from given part and part blank input BREP databases, the FEATURE_ID program uses the following logic.

1. Find a compound face occurance [Figure N-a].

    - Read in part BREP definition.
    - Scan faces and find any compound face occurrence.
        - If compound face is found then POCKET found is true and save loop ID #s associated with the compound face.

2. For each POCKET found.

    - Find edges associated with loop ID# from above. (This is the pocket opening)
    - Find loops associated with each edge found above. (These loops identify the pocket sides)
    - Find bottom loop. (This loop identifies the bottom of the pocket)
        - Load all edges associated with side loops
        - Eliminate any common edge definitions
        - Eliminate pocket opening edge definitions
        - Remaining edge definitions define the bottom loop.

3. For all SIDE and BOTTOM loops of each POCKET, define the SIDE and BOTTOM feature components of the POCKET feature.

    - Find Faces associated with each side loop found above.
    - Store all BREP data associated with each SIDE face identified above.
    - Find Face associated with bottom loop found above.
    - Store all BREP data associated with each BOTTOM face identified above.

The result of the POCKET analysis provides the definition of the POCKET features associated with a given

part and part blank in terms of two types of components. The first component or SIDE defines a side associated with a particular pocket. The second component or BOTTOM defines the bottom of the POCKET. Each POCKET must contain at least three sides and only one bottom.

APPENDIX O (POCKET FEATURE IDENTIFICATION)

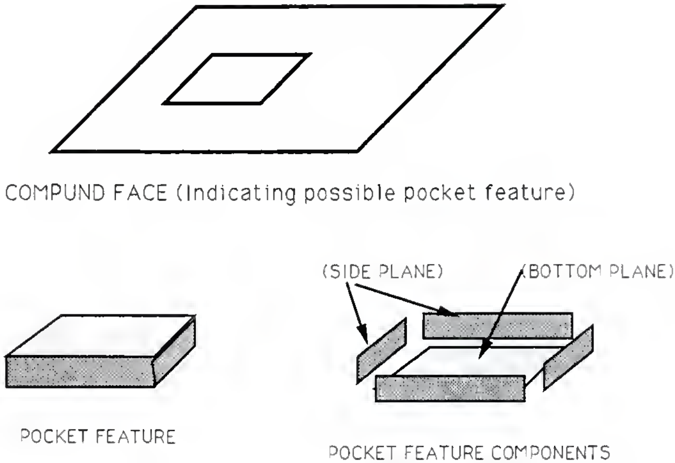


FIGURE N-a - Components of the POCKET feature.

FEATURE_ID: A PROTOTYPE SYSTEM FOR AUTOMATIC IDENTIFICATION OF NC MACHINABLE FEATURES FROM SOLID PART MODELS

by

Jeffrey A Silkman

B.S. Industrial Engineering,
Kansas State University, 1986

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

CAD/CAM systems designed for use in automated NC machining operations are currently a popular research and development topic in the area of design and manufacturing engineering. The goal of such systems is to automate data communication between the design, process planning, and NC program generation functions involved in producing NC machined parts.

The primary work presented in this paper is the development and documentation of a prototype system called FEATURE_ID. FEATURE_ID is a computer based system which is designed to provide an automated communication link between the CAD and CAM components of a CAD/CAM system limited to NC Program Development. The system has the capability of interpreting simple solid model part design databases and identifying volume of removal features needed in process planning and NC program generation functions.